Baloui

Commodore PLUS/4 Tips & Tricks

EIN DATA BECKER BUCH

Baloui

Commodore PLUS/4

Tips & Tricks

EIN DATA BECKER BUCH

ISBN 3-89011-203-X

Vorwort

Sehr geehrter Leser,

dieses Buch wendet sich an alle Besitzer eines Commodore PLUS/4. Dieser Computer ist aufgrund seines sehr guten Preis/Leistungsverhältnisses hervorragend zum "Hineinschnuppern" in die EDV geeignet, für die ersten "Gehversuche".

Der PLUS/4 bietet für seinen Preis ein sehr mächtiges BASIC und ausgezeichnete Graphikmöglichkeiten. In bezug auf das eingebaute BASIC ist er sogar seinem verbreiteteren Bruder, dem C64, deutlich überlegen und daher für erste Programmierversuche wesentlich besser geeignet.

Einem PLUS/4-Besitzer bleibt momentan auch keine andere Wahl als die Entwicklung eigener Programme, will er die Möglichkeiten seines Rechners ausnutzen. Die Lage auf dem Software-Sektor ist derzeit sehr trostlos: Außer einigen wenigen Programmlistings in Fachzeitschriften sind keinerlei Programme für diesen Rechner erhältlich, weder die gängigen Standardanwendungen wie Textverarbeitung und Dateiverwaltung noch Videospiele, Mal- oder sonstige Programme.

Das vorliegende Buch soll zum einen diesen Mangel beseitigen. Es wird eine Vielzahl von Programmen vorgestellt, die nach der Eingabe sofort voll lauffähig sind. Diese Programme stammen aus den unterschiedlichsten Bereichen (Textverarbeitung, Dateiverwaltung, Mathematik, Graphik, Elektrotechnik, Hilfsprogramme etc.), so daß hoffentlich für jeden Leser ausreichend "Futter" für seinen PLUS/4 vorhanden ist.

Zum anderen soll dieses Buch dem Leser den Einstieg in die Programmierung des PLUS/4 erleichtern. Alle Programme wurden so übersichtlich wie möglich gestaltet. Da alle Listings sehr ausführlich kommentiert sind - sowohl Programmbedienung als auch die Programmablauflogik werden erläutert - hoffe ich, daß dieses Buch nicht nur eine Programmsammlung darstellt, sondern

dem Leser Hilfestellung bei der eigenen Programmierung seines PLUS/4 bietet.

Den fortgeschrittenen Leser dürften vor allem die vielen kleinen "Tricks" interessieren, die in den Programmen verwendet werden, und die dazu beitragen, die Fähigkeiten des PLUS/4 auch in der höheren Programmiersprache BASIC voll auszunutzen (Cursor setzen, Tastendauerfunktion, simulierter Direktmodus, "Mergen" von Programmen etc.). Auf diese Tricks wird besonders ausführlich eingegangen. Jene Tricks, die immer wieder in den verschiedensten Programmen verwendet werden, wurden zusätzlich in einem eigenen Kapitel zusammengestellt und dort gesondert erläutert.

Speziell für den mit BASIC bereits ein wenig vertrauten Leser wurde ein Kapitel über die Bedienung des im PLUS/4 eingebauten Maschinensprachemonitors in dieses Buch aufgenommen, da im Handbuch leider nur seine Existenz erwähnt wird, jegliche Bedienungsanleitung jedoch fehlt.

Um Ihnen den Einstieg in die Maschinenspracheprogrammierung zu erleichtern, wird in weiteren Kapiteln auf die wichtigsten Zeropageadressen des PLUS/4, auf die freien Speicherstellen, die Speicherbelegung, die Einbindung von Maschinenroutinen in BASIC-Programme eingegangen und die wichtigsten Routinen des Betriebssystems und des BASIC-Interpreters eingegangen.

Ich danke dem "Mathematikspezialisten" V. Bühn, der mir freundlicherweise drei seiner Programme für dieses Buch zur Verfügung stellte.

Ludwigshafen, Juni 1986 Said Baloui

Achtung: *Bitte beachten Sie, daß in diesem Buch aus drucktechnischen Gründen der Hochpfeil als "^" dargestellt wird!*

Inhaltsverzeichnis

1. Einleitung

Die in diesem Buch vorgestellten Programme sind in mehrere Gruppen unterteilt (Graphik, Mathematik etc.). Zuerst werden jeweils die Eigenschaften eines Programms kurz erläutert. Die Möglichkeiten, die es bietet, und die Grenzen des Einsatzes werden dargestellt.

Im Anschluß daran wird die Programmbedienung beschrieben. Diese "Bedienungsanleitung" kann im Rahmen dieses Buches verständlicherweise nicht derart ausführlich sein, wie es bei käuflichen Programmen der Fall ist, der Leser wird jedoch in die Lage versetzt, mit dem Programm zu arbeiten und vertraut zu werden.

Nach der Bedienung wird das eigentliche Programm in Form eines Listings vorgestellt, das prinzipiell abtippfertig ist und von Ihnen sofort eingegeben werden kann. Die Eingabe bereitet keine Schwierigkeiten, da es sich meist um reine BASIC-Programme handelt, und auch Maschinenspracheprogramme in Form sogenannter "Data-Zeilen" in BASIC-Programme eingebunden sind.

Den Abschluß der Programmvorstellung bildet eine eingehende Erläuterung des Programmablaufs, die vor allem für die angehenden "Selbstprogrammierer" unter den PLUS/4-Besitzern interessant sein dürfte. Im Rahmen dieser Erläuterung wird auf die wichtigsten im Programm verwendeten Variablen und Unterprogramme eingegangen, und es werden die verwendeten Algorithmen erklärt (Algorithmus = endliche Folge von Schritten zur Lösung eines Problems).

Tricks wie zum Beispiel der simulierte Direktmodus werden ausführlich erläutert. Sollten dennoch Verständnisschwierigkeiten auftreten, empfehle ich Ihnen die Lektüre jenes Kapitels, in dem die häufig verwendeten programmtechnischen Feinheiten gesondert erläutert werden.

Viele der vorgestellten Programme verwenden Dateien zur dauerhaften Sicherung von Informationen. Die Handhabung dieser Dateien wurde speziell auf das Diskettenlaufwerk zugeschnitten, das vom Großteil der PLUS/4-Besitzer verwendet wird. Ein eigenes Kapitel erläutert die notwendigen Änderungen, die Besitzer einer Datasette zur Anpassung vornehmen müssen. Eine solche Programmumstellung ist völlig unproblematisch, wenn Sie sich an die in dem erwähnten Kapitel gegebenen Hinweise halten.

2. Hinweise zur Eingabe der Programme

In allen Programmen befinden sich sogenannte "Kommentarzeilen", die keinerlei Einfluß auf den Programmablauf haben, sondern nur dem besseren Verständnis dienen.

```
100 ...
110 for i=1 to 10
120 gosub 210:rem geschoss bewegen
130 next i
140 :
150 for i=1 to 5
160 print i
170 next i
180 ...
190 ...
210 rem ---geschoss bewegen---
220 for i=1 to 20
230 ...
```

Obwohl diese Kommentarzeilen keine Auswirkungen auf den Programmablauf haben, müssen Sie in vielen Fällen unbedingt mit eingegeben werden. Im Beispiel beginnt ab Zeile 210 ein Unterprogramm, das vom übergeordneten Programm mit dem Befehl "gosub 210" aufgerufen wird. Wenn Zeile 210 nicht eingetippt wird, erhalten Sie nach dem Start des Programms und Abarbeitung von "gosub 210" die Fehlermeldung "undefined statement error", der besagt, daß der Sprungbefehl nicht ausgeführt werden kann, da der BASIC-Interpreter das Ziel des Sprungs nicht findet, eben Zeile 210.

Man muß daher zwischen reinen Kommentarzeilen (Zeile 210) und Kommentaren unterscheiden, die sich in der gleichen Zeile wie gültige Programmbefehle befinden (Zeile 120). Erstere sollten Sie unbedingt eingeben, da sie Sprungziele darstellen können, während Sie letztere bei der Eingabe weglassen können, wenn Sie Tipparbeit sparen wollen.

Gleiches wie für reine Kommentarzeilen gilt für Zeilen, die nur aus einem Doppelpunkt bestehen (Zeile 140). Sie dienen der optischen Strukturierung, der Trennung verschiedener Programmblöcke. Zeilen, die nur aus diesem Zeichen bestehen, haben ebensowenig Einfluß auf den Programmablauf wie reine Kommentarzeilen, können jedoch ebenso wie diese Sprungziele darstellen und müssen daher mit übernommen werden.

In allen Programmen wurde Wert darauf gelegt, Kommentarzeilen oder Zeilen, die nur der Strukturierung dienen, nicht als Sprungziele zu verwenden. Dennoch sollten Sie sich die Mühe machen, auch diese Zeilen zu übernehmen. Sie erhalten dadurch weit übersichtlichere Programme. Sie sollten es sich zur Gewohnheit machen, auch bei eigenen Programmen die erwähnten Strukturierungsmöglichkeiten einzusetzen. Der reichlich bemessene Speicher des PLUS/4 bietet ausreichend Platz für gut dokumentierte Programme. Sollten Ihre eigenen Programme dennoch einmal zu umfangreich werden: In einem späteren Kapitel wird ein sogenannter "REM-Killer" vorgestellt, ein Programm, das alle reinen Kommentarzeilen und alle Zeilen, die nur aus einem Doppelpunkt bestehen, entfernt.

Schreiben Sie Ihre Programme daher mit umfangreichen Kommentaren, verwenden Sie diese Zeilen jedoch nie als Sprungziele. Wenn das Programm fertiggestellt ist, verwenden Sie den REM-Killer, um das Programm circa 25-35 Prozent (je nach Umfang der Kommentare) zu kürzen. Dieses gekürzte Programm läßt sich schneller laden und arbeitet zudem erheblich schneller als die Originalversion. Sollten Sie nachträglich Fehler entdecken, können Sie diese problemlos im strukturierten, verständlichen Originalprogramm beheben, um dieses anschließend wieder mit dem REM-Killer zu bearbeiten.

Zwei Besonderheiten, die in fast allen Listings dieses Buches vorkommen, will ich noch erkläutern, bevor wir uns mit den ersten Programmen beschäftigen.

Einbindung von Steuerzeichen

In fast allen Programmen werden sogenannte "Steuerzeichen" verwendet, mit deren Hilfe in einem laufenden Programm der Cursor bewegt werden kann, ebenso wie es im Direktmodus mit den Cursortasten möglich ist, oder mit denen innerhalb eines Programms der Revers-Modus ein- beziehungsweise ausgeschaltet werden kann.

Bei der üblichen Verwendung von Steuerzeichen erscheinen sie in Programmlistings in Form von Graphikzeichen, was die Verständlichkeit eines Programms sicher nicht gerade fördert. Meine Methode zur Einbindung von Steuerzeichen in Programme besteht darin, am Programmanfang Variablen zu deklarieren, die die benötigten Steuerzeichen enthalten, und diesen Variablen sinnvolle Namen zu geben.

Beispiele:

```
1. cu$=chr$(17):rem cursor nach unten
2. ho$=chr$(19):rem cursor nach rechts
3. ein$=chr$(18):rem revers-modus ein
```

Im Programm werden anschließend diese Variablen verwendet, deren Funktion Sie anhand des Variablennamens problemlos erkennen können.

Variablenwert ungleich null?

Sehr häufig sollen in Programmen Befehle in Abhängigkeit vom Wert einer bestimmten Variablen ausgeführt werden. Zum Beispiel sollen oftmals Befehle nur dann ausgeführt werden, wenn eine bestimmte Variable einen beliebigen Wert außer null besitzt. Dieses Problem wird üblicherweise mit folgender Konstruktion gelöst:

```
if a<>0 then ....(Befehl)
```

Die gleiche Funktion besitzt auch die folgende Konstruktion, die in den Listings dieses Buches verwendet wird:

```
if a then ....(Befehl)
```

Die Bedingung "if a" ist genau dann erfüllt, wenn "a" einen beliebigen Wert ungleich null besitzt. Beide Konstruktionen sind somit von ihrer Funktion her völlig äquivalent, die von mir bevorzugte Konstruktion besitzt jedoch den Vorteil, daß die Programme kürzer und vor allem schneller werden, wenn diese "if"-Bedingung häufig - zum Beispiel innerhalb einer Schleife - abgearbeitet werden muß.

Wenn in den folgenden Programmen sonstige spezielle "Tricks" vorkommen, werde ich diese eingehend erläutern. Ich wünsche Ihnen nun viel Erfolg beim Eingeben - und vor allem beim Arbeiten - mit den folgenden Programmen.

3. Verwendung der Datasette

Viele der in diesem Buch beschriebenen Programme verwenden Dateien zum Abspeichern von Daten. In allen Fällen wird dabei von der Verwendung eines Diskettenlaufwerks ausgegangen, dem in Verbindung mit dem PLUS/4 wohl gebräuchlichsten Massenspeicher.

Sollten Sie zu den Besitzern Datasette gehören, interessiert es Sie bestimmt, welche Programmänderungen nötig sind, wenn diese Dateien auf Band geschrieben beziehungsweise von dort gelesen werden sollen.

Die benötigten Änderungen besitzen zum Glück nur sehr geringen Umfang und sind in jedem Programm gleich. Sie beziehen sich nur auf das Öffnen einer Datei, die eigentlichen Schreib- und Lesevorgänge bleiben unverändert.

Alle in diesem Buch vorgestellten Programme verwenden beim Öffnen einer Datei einen Dateinamen, der entweder unmittelbar im Programm steht oder aber zuvor vom Benutzer eingegeben werden muß. Beim Schreiben einer Datei bestehen folgende Möglichkeiten zum Öffnen der Datei:

1. Dateiname im Programm:

```
100 open 1,8,2,"test,s,w"
```

2. Dateiname vom Benutzer eingegeben:

```
100 input"dateiname";n$
110 open 1,8,2,n$+",s,w"
```

Bei Verwendung der Floppy ist zu beachten, daß das Öffnen einer Datei, in die Daten geschrieben werden sollen und die bereits unter diesem Namen auf der Diskette existiert, zu einem

Fehler führt. Soll eine Datei überschrieben werden, muß das Sonderzeichen "Klammeraffe" dem eigentlichen Dateinamen vorausgehen, gefolgt von einem Doppelpunkt.

Verwenden Sie diese im Handbuch zur Floppy beschriebene Methode bitte nicht! Unter bestimmten Voraussetzungen arbeitet dieser Befehl fehlerhaft.

Die sicherste und in den vorgestellten Programmen verwendete Methode besteht darin, vor dem Überschreiben einer Datei diese zu löschen. Das Löschen einer Datei ist mit folgender Befehlsfolge (nur im Programm!) möglich:

1. Dateiname im Programm:

```
100 open 15,8,15,"s:test":close 15:open 1,8,2,"test,s,w"
```

2. Dateiname vom Benutzer eingegeben:

```
100 input"dateiname";n$
110 open 15,8,15,"s:"+n$:close 15:open 1,8,2,n$+",s,w"
```

Dieser "Vorspann" vor dem eigentlichen Befehl zum Öffnen der Datei muß bei Verwendung der Datasette entfernt werden!

Weitere Unterschiede bestehen in der beim "OPEN"-Befehl angegebenen Geräteadresse (Datasette: eins statt acht), der angegebenen Sekundäradresse (Datasette: Lesen: null; Schreiben: eins), und dem bei Verwendung der Datasette nicht benötigten Zusatz ",s,w" beziehungsweise ",s,r", der entfernt werden muß. Die beiden vorgestellten Beispiele zum Schreiben einer Datei auf Diskette können Sie auf folgende Weise für die Arbeit mit der Datasette umsetzen:

1. Dateiname im Programm:

```
100 open 1,1,1,"test"
```

2. Dateiname vom Benutzer eingegeben:

```
100 input"dateiname";n$
110 open 1,1,1,n$
```

Beim Lesen einer Datei ist es nicht nötig, eine Datei zu löschen, es muß jedoch ebenfalls auf eine geänderte Syntax des "OPEN"-Befehls geachtet werden. Zum Lesen einer Datei von Diskette werden in den vorgestellten Programmen folgende Befehle verwendet:

1. Dateiname im Programm:

```
100 open 1,8,2,"test,s,r"
```

2. Dateiname vom Benutzer eingegeben:

```
100 input"dateiname";n$
110 open 1,8,2,n$+",s,r"
```

Die Umsetzung dieser beiden Varianten für die Zusammenarbeit mit der Datasette:

1. Dateiname im Programm:

```
100 open 1,1,0,"test"
```

2. Dateiname vom Benutzer eingegeben:

```
100 input"dateiname";n$
110 open 1,1,0,n$
```

Weitere Änderungen sind nicht erforderlich. Probieren Sie die Umstellung auf Datasette am besten mit dem einfachsten Programm aus, das Dateien verwendet, dem Programm "KFZ".

4. Anwenderprogramme

In den folgenden Kapiteln werden Programme aus den verschiedensten Bereichen vorgestellt. Alle Programme wurden eingehend getestet. Versuchen Sie bitte nicht, Programmzeilen zu ändern, die Ihnen unverständlich erscheinen. Tippen Sie die Listings wie vorliegend ab.

4.1. Spiele

Vier Spiele werde ich vorstellen, die sehr unterschiedlich sind. Die Spanne reicht von dem folgenden Actionspiel "Schiffe versenken" bis zu "ASW" und "Conway's Life", Programmen, die sich keineswegs eindeutig unter die Rubrik "Spiele" einordnen lassen.

4.1.1. Schiffe versenken

Dieses Spiel ist ein Videospiel im Textmodus, das heißt nicht in hochauflösender Graphik. Unter Verwendung der Graphikzeichen des PLUS/4 wird ein Schiff gemalt, das sich von links nach rechts über den Bildschirm bewegt. Aufgabe des Benutzers, dem eine ebenfalls mit Graphikzeichen dargestellte Kanone zur Verfügung steht, ist es, das Schiff "abzuschiessen", bevor es den rechten Bildschirmrand erreicht.

Wie Sie bei der Benutzung sehen werden, ist dieses Spiel - trotz Kriegsschiff und Kanone - weniger ein brutales Kriegsspiel, sondern viel eher ein Test für Ihr Reaktions- und Schätzvermögen.

Programmbedienung

Nach dem Starten will das Programm von Ihnen wissen, mit welcher Geschwindigkeit Sie spielen wollen. Sie müssen eine Zahl zwischen null und neun eingeben (Betätigung der Zahlen-

taste genügt, eine Bestätigung durch "RETURN" ist überflüssig). Je niedriger die von Ihnen eingegebene Zahl ist, desto schneller bewegen sich sowohl Schiff als auch Kanonenkugel.

Das Programm fragt Sie nun, ob Sie Anfänger oder Fortgeschrittener sind. Betätigen Sie entweder die Taste "a" (Anfänger) oder die Taste "f" (Fortgeschrittener).

Wenn Sie beide Fragen beantwortet haben, erscheint auf dem Bildschirm unverzüglich ein Schiff am linken Bildschirmrand. Ihre Kanone ist unbeweglich, das heißt Sie müssen exakt abschätzen, wann Sie die Kanonenkugel abschießen sollten, damit sich die Bahnen von Schiff und Kugel kreuzen. Das Abschießen der Kanonenkugel erfolgt durch Drücken der Taste "RETURN".

Am unteren Bildschirmrand wird Ihnen die jeweils aktuelle Trefferanzahl angezeigt. Ein Treffer erhöht diesen Zählerstand um einen Punkt. Jede Runde besteht aus zehn Durchläufen, daher sind maximal zehn Punkte zu erreichen.

Nach zehn Durchläufen werden Sie gefragt, ob Sie eine weitere Runde wünschen. Eine neue Runde wird durch Betätigung von "j" gestartet.

Um Ihren Ehrgeiz anzuspornen: Mein persönlicher Rekord steht auf zehn Treffern im Anfängermodus beziehungsweise neun Treffern im Fortgeschrittenenmodus (jeweils mit maximaler Geschwindigkeit). Und nun wünsche ich Ihnen viel Spaß und möglichst viele versenkte Schiffe.

Programmlisting "Schiffe versenken"

```
100 rem -----schiffe versenken-----
110 rem --(januar 1986/s.baloui)---
120 :
130 rem ---variablen---
140 rem g=geschwindigkeit
150 rem ho$=steuerz. cursor home
160 rem ein$=steuerz. revers ein
```

```
170 rem aus$=steuerz. revers aus
180 rem ss=spalte des schiffs
190 rem rz=randomzeile (zufaellig ermittelt), in der sich
das schiff bewegt
200 rem gz=geschosszeile
210 rem tr=trefferanzahl
220 rem flag: 0/1=ruhendes/bewegtes geschoss
230 rem tflag: 0/1=kein treffer/treffer
240 ho$=chr$(19)
250 ein$=chr$(18)
260 aus$=chr$(146)
270 :
280 rem ---infos+initialisierung---
290 print chr$(142):scnclr
300 print "geschwindigkeit (0-9) ?"
310 getkey a$:g=val(a$)
320 print ho$;
330 print "anfaenger (a)/fortgeschrittener (f) ?"
340 getkey m$
350 :
360 print ho$;
370 print "schiffe versenken ('return' = feuern)
380 tr=0
390 s$=" "+chr$(127)+ein$+"  "+aus$+chr$(169):rem schiff
400 :
410 :
420 rem -----hauptschleife-----
430 for i=1 to 10
440 char 0,0,24,"treffer"+str$(tr)+" "
450 char 0,24,24,ein$+"   "+aus$
460 ss=1:rem schiff: startspalte
470 gz=23:rem geschoss: startzeile
480 if m$="a" then rz=int(rnd(0)*17)+5:else rz=int(rnd(0)*
10)+2
490 :
500 rem ---innere schleife---
510 for ii=1 to 35
520 char 0,ii,rz,s$:rem schiff malen
530 :
540 for z=1 to g*2
```

```
550 get cr$:rem 'return'?
560 if cr$=chr$(13) then flag=1
570 next z
580 :
590 if flag then gosub 760:rem geschoss bewegen
600 if gz<rz or tflag then 640
610 gosub 820:rem treffer abfragen
620 next ii
630 :
640 char 0,ii,rz,"     "
650 tflag=0
660 flag=0
670 char 0,25,gz," "
680 next i
690 :
700 char 0,0,24,"treffer:"+str$(tr)+"   noch eine runde
(j/n) ?"
710 getkey a$
720 if a$="j" then run:else end
730 :
740 :
750 rem ---geschoss bewegen---
760 char 0,25,gz," ":rem geschoss loeschen
770 gz=gz-1:rem zeile dekrementieren
780 char 0,25,gz,chr$(113):rem schiff malen
790 return
800 :
810 rem ---treffer abfragen---
820 if flag=0 or gz<>rz or ii<21 or ii>24 then return:rem
kein treffer?
830 tr=tr+1:rem trefferzahl inkrementieren
840 char 0,8,24,str$(tr)
850 tflag=1:rem treffer merken
860 return
```

Programmablauf

Den eigentlichen Programmbeginn bildet die Definition mehrerer Variablen, denen die Steuerzeichen für "Cursor home", "Revers on" und "Revers off" zugewiesen werden.

Die Zeilen 280 bis 390 dienen der Initialisierung mehrerer Variablen, das heißt der Festlegung von Parametern, die den Spielablauf bestimmen. In der Variablen "g" wird die vom Benutzer eingegebene Geschwindigkeit gespeichert (0-9) und in der Variablen "m$" der Modus (m$="a" oder m$="f"). Der Cursor wird auf die erste Bildschirmzeile gesetzt und der Benutzer erhält Informationen zum Abfeuern der Kanone (Zeilen 360-370). Die Variable "tr", die im weiteren Spielverlauf die aktuelle Trefferanzahl enthält, erhält den Wert null und in "s$" werden die Graphikzeichen gespeichert, die das Schiff symbolisieren sollen.

Das Hauptprogramm ist in eine Schleife eingebettet, die vom Anfangswert eins bis zum Endwert zehn durchlaufen wird (zehn Durchgänge).

Zu Beginn eines Durchgangs wird in Zeile 24 die momentane Trefferanzahl "tr" ausgegeben und die Kanone gezeichnet (drei inverse Spaces).

Danach wird die Variable "ss" (Spalte, in der sich das Schiff befindet) auf den Wert eins gesetzt, da sich das Schiff am linken Bildschirmrand befindet, und die Variable "gz" (Zeile, in der sich das Geschoß befindet) erhält den Wert 23 (das Geschoß befindet sich noch am unteren Bildschirmrand).

Wurde der Anfängermodus gewählt (m$="a"), wird nun eine Zufallszahl "rz" zwischen 5 und 22 ermittelt, sonst eine Zufallszahl im Bereich zwischen zwei und zwölf. Diese Zufallszahl bestimmt die Bildschirmzeile, in der sich das Schiff bewegt, je nach Modus mehr im oberen oder mehr im unteren Bildschirmbereich.

Nun beginnt eine weitere Schleife, deren Schleifenvariable "ii" (1-35) der Bildschirmspalte entspricht, in der sich das Schiff befindet. In Zeile 520 wird das Schiff an der durch "ii" und "rz" festgelegten Bildschirmposition gezeichnet.

In der folgenden Schleife (Zeilen 540-570) wird die Tastatur abgefragt. Wurde die Taste "RETURN" gedrückt, wird also das Geschoß abgefeuert, erhält "flag" den Wert eins. Diese Schleife dient gleichzeitig als Verzögerungsschleife (die Anzahl der Schleifendurchgänge hängt von der Geschwindigkeit "g" ab).

Wenn "flag" den Wert eins besitzt, das Geschoß abgefeuert wurde, wird anschließend ein Unterprogramm aufgerufen, das dieses Geschoß um eine Zeile nach oben bewegt.

Wenn "gz" (Geschoßzeile) kleiner ist als "rz" (Schiffs-zeile), befindet sich das Geschoß somit bereits oberhalb des Schiffs. Damit ist ein Treffer nicht mehr möglich, und der momentane Durchgang wird beendet. Der Durchgang wird auch beendet, wenn "tflag" einen Wert ungleich null besitzt. "tflag" erhält im Unterprogramm zur Trefferabfrage den Wert eins, wenn ein Treffer erfolgte. Dieses Unterprogramm wird zum Schluß der inneren Schleife angesprungen.

Nach Beenden eines Durchgangs werden sowohl das Schiff als auch ein eventuell gestartetes Geschoß gelöscht, "tflag" (Treffer: ja/nein) und "flag" (Geschoß abgefeuert: ja/nein) werden auf null gesetzt, und der nächste Durchgang beginnt (Zeilen 640-680).

Nach zehn Durchgängen wird das Programm neu gestartet, wenn der Benutzer die entsprechende Frage mit "j" beantwortet (Zeilen 700-720).

Die Zeilen 750-790 bilden ein Unterprogramm, das bei jedem Aufruf das Geschoß um eine Zeile nach oben bewegt. Zuerst wird das Geschoß an der momentanen Position gelöscht (Spalte 25, Zeile "gz"), die Geschoßzeile danach dekrementiert (um eins vermindert), und zum Abschluß wird das Geschoß an der neuen Position gezeichnet.

Die Zeilen 810-860 bilden die Trefferabfrage. Ein Treffer erfolgte genau dann, wenn die Positionen von Schiff und Geschoß übereinstimmen, das heißt, wenn die Geschoßzeile "gz" mit der Schiffszeile "rz" übereinstimmt und sich die Schiffsspalte im Bereich zwischen 21 und 24 bewegt. Bedenken Sie, daß die Geschoßspalte zwar unverändert festgelegt ist (Spalte 25), das Schiffssymbol sich jedoch über mehrere Bildschirmspalten erstreckt.

Erfolgte kein Treffer, wird das Unterprogramm unverzüglich verlassen, ansonsten wird zuvor die Trefferanzahl "tr" erhöht, die neue Punktzahl ausgegeben und "tflag" erhält den Wert eins, um dem Hauptprogramm anzuzeigen, daß ein Treffer erfolgte.

4.1.2. Reaktionstest

Das folgende Programm testet Ihr Reaktionsvermögen. Ihnen wird die Aufgabe gestellt, auf Aufforderung eine beliebige Taste zu betätigen und das Programm mißt die von Ihnen benötigte Reaktionszeit.

Ich empfehle jedem Leser, der an der Programmierung des PLUS/4 interessiert ist, die Erläuterung des Programmablaufs zu studieren. Im betreffenden Abschnitt wird die Verwendung des Tastaturpuffers des PLUS/4 erläutert, ohne die manche Programme - zum Beispiel auch dieser Reaktionstest - nicht erstellt werden können.

Programmbedienung

Die Programmbedienung ist außerordentlich einfach: Das Programm teilt Ihnen mit, daß Sie auf Aufforderung eine beliebige Taste - bitte nicht die Taste "STOP" - drücken sollen und teilt Ihnen die zwischen der Aufforderung und Ihrer Reaktion vergangene Zeit mit, Ihre Reaktionszeit.

Nach zehn Durchgängen wird Ihnen Ihre durchschnittliche Reaktionszeit mitgeteilt und Sie werden gefragt, ob Sie weitere zehn Durchgänge wünschen. Wenn Sie mit "j" für "ja" antworten, beginnt eine neue Runde.

Programmlisting

```
100 rem -----reaktionstest-----
110 rem -(januar 1986/baloui)--
120 :
130 rem ---variablen---
140 rem zo=zeit old (messbeginn)
150 rem zn=zeit new (messende)
160 rem rz=reaktionszeit in sekunden
170 rem sr=summe der reaktionszeiten
180 rem tz=zaehler: anzahl zeichen im tastaturpuffer
190 :
200 tz=239
210 printchr$(14):rem gross/klein
220 scnclr
230 print tab(13)"Reaktionstest"
240 print:print
250 print " Bitte auf Aufforderung Taste druecken"
260 :
270 for i=1 to 2000:next
280 for i=1 to 10
290 scnclr
300 a=int(rnd(0)*2000)+500
310 for ii=1 to a:next
320 poke tz,0:rem tastatuspufferzaehler init.
330 print tab(17)"Taste"
340 zo=ti
350 getkey a$
360 zn=ti
370 rz=(zn-zo)/60:sr=sr+rz
380 print:print"    Reaktionszeit:"rz"sec"
390 for ii=1 to 1000:next
400 next i
410 :
```

```
420 print:print
430 print"Durchschnitt:"sr/10"sec"
440 print:print
450 print"Noch eine Runde (j/n) ?"
460 getkey a$
470 if a$="j" then run
```

Programmablauf

Zum Verständniss des Programmablaufs ist es wichtig zu wissen, daß der PLUS/4 über eine interne Uhr verfügt und die Uhrzeit mit Hilfe der eigens für diesen Zweck reservierten Variablen "ti" abgefragt werden kann. Der Wert von "ti" wird jede sechzigstel Sekunde inkrementiert, das heißt um eins erhöht. Die Zeitmessung beginnt daher mit dem Speichern des momentanen Wertes von "ti", der in eine andere Variable - im vorliegenden Programm in "zo" - übertragen wird.

Die verstrichene Zeit ergibt sich daher aus dem Wert von "ti" zu einem späteren Zeitpunkt und dem Wert "zo" zu Beginn der Messung. Die ermittelte Differenz muß noch durch 60 geteilt werden, um die Zeitdifferenz in Sekunden zu erhalten.

In den Zeilen 200-250 werden dem Benutzer Informationen zur Programmbedienung gegeben. Zeile 270 stellt eine Warteschleife dar, nach deren Ablauf der eigentliche Reaktionstest beginnt. Eine Runde besteht aus zehn Durchläufen, der Reaktionstest ist daher in eine Schleife eingebettet, die von eins bis zehn läuft (Zeile 280).

In jedem Durchgang wird zuerst der Bildschirm gelöscht und anschließend eine Zufallszahl im Bereich zwischen 500 und 2500 ermittelt (Zeile 300). Diese Zufallszahl bestimmt die Anzahl der Durchgänge einer zweiten Warteschleife (Zeile 310).

Nach Ablauf dieser zufällig verstrichenen Zeitspanne beginnt die Messung. Der Benutzer wird aufgefordert, eine beliebige Taste zu drücken. Der momentane Inhalt von "ti" wird in die Variable "zo" (zeit old=Zeit bei Messbeginn) übertragen. Nachdem eine

beliebige Taste gedrückt wurde, wird der neue Inhalt von "ti" in die Variable "zn" (Zeit new=Zeit bei Messende) übertragen (Zeilen 330-360).

Die Differenz zwischen "zn" und "zo" ergibt die benötigte Reakionszeit in sechzigstel Sekunden und wird daher noch durch sechzig geteilt. Die - nun in Sekunden - ermittelte Reaktionszeit wird in der Variablen "rz" (Reaktionszeit) gespeichert. Die Variable "sr" (Summe der Reaktionszeiten) wird um den ermittelten Wert erhöht. Am Ende der zehn Durchgänge enthält "sr" somit die Summe der zehn einzelnen Reaktionszeiten, die zur Berechnung des Durchschnittswertes erforderlich ist (Zeile 370).

Die ermittelte Reaktionszeit wird dem Benutzer gemeldet, und nachdem er durch eine Warteschleife Gelegenheit zum Lesen der Meldung erhielt, beginnt der nächste Durchgang (Zeilen 380-400).

Nach Beendigung aller zehn Durchgänge wird die Summe der einzelnen Reaktionszeiten "sr" durch zehn geteilt und dem Benutzer die ermittelte durchschnittliche Reaktionszeit mitgeteilt. Anschließend wird das Programm neu gestartet, wenn er die folgende Frage mit "j" beantwortet (Zeilen 420-470).

Zwei besonders interessante Programmzeilen wurden in dieser Erläuterung des Programmablaufs noch nicht erwähnt: Die Zeilen 200 und 320. In Zeile 200 wird die Variable "tz" definiert. Sie erhält den Wert 239. In Zeile 320 wird unmittelbar vor Beginn einer Messung in diese Speicherstelle der Wert null "gepokt".

Ohne Kenntnis des Tastaturpuffers im PLUS/4 ist diese Zeile unverständlich. Sie alle werden sicherlich bereits festgestellt haben, daß der PLUS/4 Aktionen auch dann ausführt, wenn Ihre Befehle ohne ausreichenden zeitlichen Abstand gegeben werden. Vor allem in BASIC-Programmen wird diese Eigenschaft oft an einem "nachhinkenden" Cursor sichtbar. Wenn Sie zum Beispiel in einer - in BASIC programmierten - Textverarbeitung Text extrem schnell eingeben, wird das

BASIC-Programm nicht in der Lage sein, Ihre Eingaben ebenso schnell zu verarbeiten. In diesem Fall erscheinen von Ihnen eingegebene Zeichen nachträglich, wenn Sie bereits Ihre Eingabe beendet haben.

Die Begründung für diese Eigenschaft ist der sogenannte "Tastaturpuffer" des PLUS/4. In diesem Puffer werden alle eingegebenen Zeichen gespeichert. Während des Programmablaufs werden die Zeichen, die sich im Puffer befinden, gelesen und verarbeitet. Der Tastaturpuffer sorgt auf diese Weise dafür, daß unsere Eingaben nicht verloren gehen, auch wenn das jeweilige BASIC-Programm in der Verarbeitungsgeschwindigkeit mit unserem Eingabetempo nicht Schritt halten kann.

Die Erstellung des vorliegenden Programms wird durch diese Eigenschaft jedoch erheblich erschwert: Der Benutzer kann bereits vor Beginn der Messung eine Taste drücken, das heißt bevor die Aufforderung dazu erscheint. Dieser Tastendruck wird im Tastaturpuffer gespeichert, jedoch noch nicht bearbeitet. Wenn nun die eigentliche Messung beginnt und der Befehl "getkey a$" vom BASIC-Interpreter abgearbeitet wird, stellt dieser fest, daß der Tastaturpuffer ein noch nicht bearbeitetes Zeichen enthält und eine Taste gedrückt wurde.

Der Befehl "getkey a$", der auf die Betätigung einer Taste wartet, gilt damit als erfüllt und die nächste Programmzeile wird sofort abgearbeitet. Diese Eigenschaft eröffnet Möglichkeiten zum "Beschummeln" im vorliegenden Reaktionstest: Der Benutzer betätigt bereits vor Messbeginn eine beliebige Taste und erhält dadurch unweigerlich Reaktionszeiten von null Sekunden.

Zum Tastaturpuffer gehört jedoch auch ein Zähler, der die Anzahl der noch nicht abgearbeiteten Zeichen enthält. Dieser Zähler befindet sich in Speicherstelle 239. Durch den Befehl "poke tz,0" unmittelbar vor Messbeginn erhält der Zähler den Wert null, dem PLUS/4 wird dadurch mitgeteilt, daß der Tastaturpuffer keine noch nicht abgearbeiteten Zeichen enthält, die nachträglich verarbeitet werden müssen.

Wie effektiv dieser kleine Trick ist, können Sie selbst ausprobieren: Löschen Sie Zeile 320, bevor Sie das Programm starten. Drücken Sie bei den nun folgenden Durchgängen bereits eine beliebige Taste, bevor die Aufforderung dazu auf dem Bildschirm erscheint. Glauben Sie mir, Ihre Reaktionszeiten werden von nun an hervorragend sein.

4.1.3. Conway's Life

Die Einordnung von "Conway's Life" unter die Rubrik "Spiele" wird zweifellos bei vielen Lesern Widerspruch hervorrufen. Ein typisches Merkmal von Spielen, die aktive Teilnahme, ist bei diesem Programm nur in sehr geringem Maße vorhanden. Conway's Life ist die Simulation der Entwicklung einer Population von Generation zu Generation. Diese Population kann aus einer beliebigen Art von Lebewesen bestehen.

Die Population wird auf einem Spielfeld dargestellt, das aus einzelnen Zellen besteht, wobei jede dieser Zellen leer sein kann oder aber mit einem Lebewesen besetzt ist.

Die Entwicklung der jeweiligen Population - ausgehend von einem bestimmten Ausgangszustand, den der Spieler festlegt - verläuft nach folgenden Regeln:

Entstehung neuen Lebens: In einer bisher leeren Zelle entsteht ein neues Lebewesen, erfolgt eine "Geburt", wenn diese Zelle von genau drei besetzten Nachbarzellen umgeben ist. Da jede Zelle (analog zum Schachbrett) acht Nachbarzellen hat, bedeutet diese Bedingung, daß drei der Zellen besetzt und die restlichen fünf leer sein müssen.

Tod eines Lebewesen: Eine bisher besetzte Zelle wird leer, das heißt, daß ein darin existierendes Lebewesen "stirbt", wenn weniger als zwei oder aber mehr als drei Nachbarzellen Lebewesen beherbergen. Im ersten Fall stirbt das Lebewesen an "Vereinsamung"; im zweiten Fall an "Überbevölkerung".

Mit dieser Simulation können Sie als Zuschauer die Entwicklung einer von Ihnen erzeugten Population bis zu einem von zwei Endzuständen verfolgen: Entweder stirbt die Population aus, oder aber es entwickelt sich ein stabiler Zustand, der der Population ein dauerhaftes Überleben sichert.

Programmbedienung

Nach dem Starten des Programms erscheint auf dem Bildschirm das Spielfeld. Alle Zellen der Population sind unbesetzt. Ihre Aufgabe besteht nun darin, einen Ausgangszustand vorzugeben. Sie können den Cursor, der als inverser Block sichtbar ist, beliebig über den Bildschirm bewegen. Halten Sie sich jedoch bitte an das abgegrenzte Spielfeld.

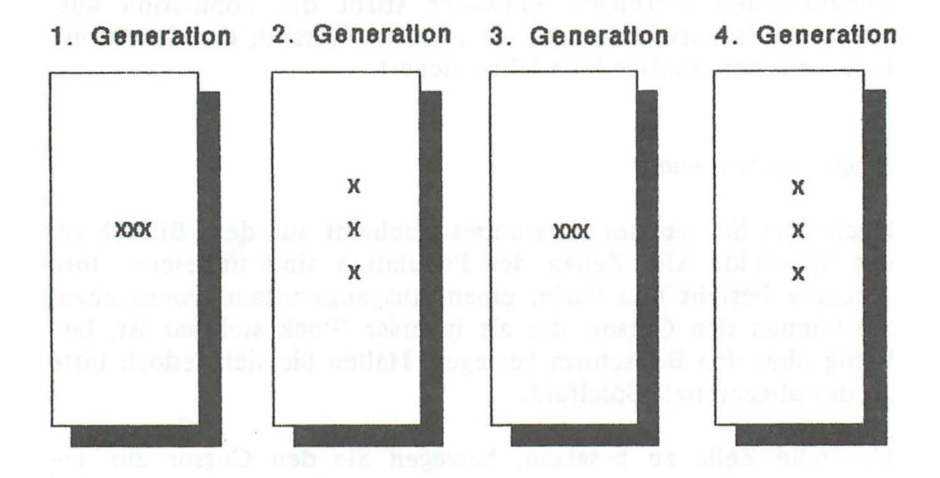
Um eine Zelle zu besetzen, bewegen Sie den Cursor zur jeweiligen Position und drücken die Taste "x". Markieren Sie auf diese Weise mehrere mit einem Lebewesen besetzte Zellen. Welchen Ausgangszustand Sie auf diese Weise für die Population herstellen, bleibt völlig Ihnen überlassen.

Wenn der von Ihnen gewünschte Ausgangszustand erstellt wurde, betätigen Sie die Taste "RETURN". Das Programm wird nun aus der von Ihnen erzeugten ersten Generation nach den geschilderten Regeln die zweite Generation ermitteln und im Spielfeld ausgeben. Aus dieser zweiten Generation ermittelt es die dritte, anschließend die vierte Generation und so weiter.

Sie können die Entwicklung "Ihrer" Population Schritt für Schritt verfolgen und bereits zu Beginn Vermutungen darüber anstellen, ob sie ausstirbt oder sich als auf Dauer überlebensfähig erweist.

Wenn keine Änderungen mehr sichtbar sind, können Sie das Programm mit der Taste "STOP" abbrechen, neu starten und einen weiteren Ausgangszustand eingeben.

Beispiel einer stabilen Struktur



Diese "Rotorstruktur" ist ein gutes Beispiel dafür, daß eine Struktur "dynamische Stabilität" besitzen kann. Sie ist stabil, jedoch nicht unveränderlich. Bei jedem Generations-wechsel dreht sich der Rotor um 90 Grad, solange, bis Sie das Programm mit der "STOP"-Taste unterbrechen. Nach den Regeln ergeben sich folgende Berechnungen für die Anzahl der Nachbarn, die jede Zelle besitzt:

1. Generation	2. Generation	3. Generation	4. Generation
00000000	00000000	00000000	00000000
00000000	00010000	00000000	00010000
00232000	00212000	00232000	00212000
01121100	00323000	01121100	00323000
00232000	00212000	00232000	00212000
00000000	00010000	00000000	00010000
00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000

Programmlisting

```
100 rem -----conway's life-----
110 rem (januar 1986/s.baloui)
120 :
130 rem ---variablenerlaeuterung---
140 rem cs=cursorspalte
150 rem cz=cursorzeile
160 rem sc=screen-startadresse
170 rem ho$=steuerz. cursor home
180 rem cl$=steuerz. cursor links
190 rem cu$=steuerz. cursor unten
200 rem z(...,...)=spielfeld (0/1=kein leben/leben)
210 rem i(...,...)=auswertungsfeld (summen umgebender leben
fuer jedes feld)
220 :
230 cs=202
240 cz=205
250 sc=3072
260 ho$=chr$(19)
270 cl$=chr$(157)
280 cu$=chr$(17)
290 :
300 print chr$(142):rem gross/graphik
```

```
310 gosub 640:rem programmstart
320 :
330 rem ---z(..,..) auswerten---
340 for i=1 to 10
350 for j=1 to 10
360 h=0
370 if z(i-1,j-1) then h=h+1
380 if z(i-1,j) then h=h+1
390 if z(i-1,j+1) then h=h+1
400 if z(i,j-1) then h=h+1
410 if z(i,j+1) then h=h+1
420 if z(i+1,j-1) then h=h+1
430 if z(i+1,j) then h=h+1
440 if z(i+1,j+1) then h=h+1
450 i(i,j)=h
460 next j,i
470 :
480 print ho$ cu$ cu$ cu$ cu$ cu$ cu$ cu$ tab(11);
490 for i=1 to 10
500 for j=1 to 10
510 h=i(i,j)
520 if h=3 then z(i,j)=1:goto 560
530 if h<2 or h>3 then z(i,j)=0
540 :
550 rem ---ausgabe---
560 if z(i,j)=1 then print"x";:goto 580
570 print" ";
580 next
590 print:print tab(11);
600 next
610 goto 330
620 :
630 rem ---init---
640 dim z(11,11),i(11,11)
650 gosub 760:print ho$ cu$ cu$ cu$ cu$ cu$ cu$ cu$ tab(11)
660 a=peek(cz)*40+peek(cs)+sc
670 poke a,peek(a) or 128
680 getkey a$
690 poke a,peek(a) and 127
700 if asc(a$)=13 then return
```

```
710 if a$="x" then z(peek(cz)-6,peek(cs)-10)=1
720 print a$;
730 goto 660
740 :
750 rem ---rahmen zeichnen---
760 scnclr
770 print tab(10)"conway's life"
780 print tab(10)"-------------"
790 print ho$ cu$ cu$ cu$ cu$ cu$
800 rem grenze
810 print tab(10)"UCCCCCCCCCCI"
820 fori=1to10:print tab(10)"B          B":next
830 print tab(10)"JCCCCCCCCCCK"
840 return
```

Ein Hinweis zur Eingabe des Programms: Schalten Sie vor Beginn der Eingabe Ihren PLUS/4 durch gleichzeitiges Drücken der Taste mit dem Commodore-Zeichen und der Shift-Taste in den sogenannten "Gross/Klein"-Modus. Bereits auf dem Bildschirm vorhandene Zeichen sollten anschließend in Kleinschrift dargestellt sein. Wenn dies nicht der Fall ist, betätigen Sie diese Tastenkombination ein zweites Mal.

In diesem Gross/Klein-Modus können Sie das Unterprogramm zum Zeichnen des Spielfeldrahmens problemlos wie im abgebildeten Listing eingeben. Die darin enthaltenen Grossbuchstaben, die Sie bitte unter gleichzeitiger Betätigung von "SHIFT" eingeben, werden nach dem Starten des Programms und dem Umschalten in den Gross/Graphik-Modus (Zeile 300) als Graphikzeichen dargestellt werden, die den Spielfeldrahmen ergeben.

Programmablauf

Am Programmanfang werden Steuerzeichen und sonstige häufig verwendete Variablen definiert. "cs" ist die Speicherstelle, in der die momentane Cursorspalte abgelegt ist (0-39), und "cz" die Speicherstelle, in der sich die aktuelle Cursorzeile befindet (0-

24). "sc" ist die Adresse, an der im RAM des PLUS/4 der Bildschirmspeicher beginnt, in dem der komplette Bildschirminhalt gespeichert ist (25 Zeilen mit je 40 Zeichen = 1000 Zeichen).

Das Spielfeld besteht aus einer 10*10 - Matrix. Benötigt werden zwei Variablenarrays: eines, um die momentane Besetzung des Spielfeldes mit leeren beziehungsweise besetzten Zellen festzuhalten, und ein weiteres, in dem die Ergebnisse der Berechnungen (Anzahl der Nachbarn jeder Zelle) gespeichert werden.

In dem ab Zeile 640 beginnenden Hauptprogramm werden zuerst diese beiden Arrays dimensioniert. Danach wird ein Unterprogramm aufgerufen, das einen Rahmen um das Spielfeld zeichnet (Zeilen 760-870).

Die Zeilen 660-730 stellen einen sogenannten "Editor" dar, der es dem Benutzer erlaubt, innerhalb eines laufenden Programms beliebig auf dem Bildschirm umherzuwandern und die Ausgangspopulation aufzubauen.

Die Art und Weise, wie dieser Editor einen Cursor auf dem Bildschirm darstellt (Zeilen 660-670), ist ohne Maschinensprachekenntnisse leider nur sehr schwer zu verstehen.

Der Cursor wird als inverses Zeichen dargestellt. Zu diesem Zweck wird die momentane Position des Cursors ermittelt (Spalte und Zeile) und mit Hilfe dieser Werte die entsprechende Stelle im Bildschirmspeicher errechnet (Zeile 660). Die ermittelte Speicherstelle enthält das Zeichen, das sich an der aktuellen Cursorposition befindet. Die diesem Zeichen zugeordnete Zahl wird mit 128 verknüpft ("ODER"-Verknüpfung). Die Zahl, die sich ergibt, entspricht dem gleichen Zeichen in inverser Darstellung. Das "Poken" dieser Zahl in die errechnete Bildschirmspeicherstelle führt dazu, daß dieses Zeichen invers dargestellt wird.

Das Programm wartet nun auf eine beliebige, vom Benutzer zu betätigende Taste. Wurde eine Taste gedrückt, wird das Zeichen an der momentanen Cursorposition mit "or 127" verknüpft. Diese Verknüpfung ergibt den Zahlenwert, der der Normaldarstellung

dieses Zeichens entspricht und wird in die errechnete Bildschirmspeicherstelle gepokt. Nach jeder Betätigung einer Taste wird das zuvor invers dargestellte Zeichen somit wieder normal dargestellt, das heißt der Cursor wird ausgeschaltet.

Wurde die Taste "RETURN" gedrückt (ASCII-Code 13), ist die Aufgabe des Editors beendet und das Unterprogramm wird verlassen (Zeile 700). Wurde eine andere Taste betätigt, wird untersucht, ob es eine Taste zur Bewegung des Cursors war oder aber die Taste "x" gedrückt und damit eine Zelle des Feldes besetzt wurde. Soll eine Zelle besetzt werden, wird in die entsprechende Spalten- beziehungsweise Zeilenposition des zugeordneten Arrays "z(...,...)" der Wert eins eingetragen. Dieser Wert kennzeichnet eine besetzte Zelle.

Anschließend wird das eingegebene Zeichen ("a$") ausgegeben und damit entweder der Cursor bewegt (wenn eine Cursortaste betätigt wurde), oder aber ein "x" an der jeweiligen Spiel-feldposition gezeichnet (Zeile ein Sprung 720).

Wurde die Taste "RETURN" betätigt, folgt ab Zeile 330 die eigentliche Auswertung der Generationsfolgen, die wesentlich leichter zu verstehen ist als die Funktionsweise des Editors.

Das Array "z(...,...)" enthält die Zellenbesetzungen. Überall dort, wo vom Benutzer ein "x" eingegeben wurde, enthält dieses Array den Wert eins zum Zeichen einer besetzten Zelle. Leere Zellen werden in dieser zweidimensionalen Matrix durch den Wert null gekennzeichnet.

Zur Auswertung werden zwei ineinander geschachtelte Schleifen benötigt, um nacheinander alle Spalten und Zeilen des Arrays "z(...,...)" abzuarbeiten. Da das Spielfeld eine 10*10-Matrix darstellt, laufen die Schleifenindizes jeweils von eins bis zehn. In dem Array werden daher nacheinander die Spalten eins bis zehn der ersten Zeile, die Spalten eins bis zehn der zweiten Zeile, und zum Schluß alle Spalten der zehnten Zeile abgearbeitet.

Während der Abarbeitung einer Position des Arrays werden alle benachbarten Positionen auf den Wert eins hin untersucht. Wenn

zum Beispiel "i" den Wert drei und "j" den Wert fünf besitzt (Zeile drei, Spalte fünf), müssen folgende Arraypositionen untersucht werden:

```
z(2,4), z(2,5), z(2,6), z(3,4),
z(3,6), z(4,4), z(4,5), z(4,6)
```

Immer dann, wenn in einer der untersuchten Nachbarzellen Leben festgestellt wurde, wird die Variable "h" um eins erhöht. Wurden alle Nachbarn einer Zelle untersucht, enthält "h" daher die Anzahl der besetzten Nachbarzellen. Diese Zahl wird in die entsprechende Position im Array "i(...,...)" eingetragen. Wenn zum Beispiel die Zelle "z(3,5)" von drei besetzten Nachbarzellen umgeben ist, wird im entsprechenden Feld des Arrays "i(..,...)" der Wert drei eingetragen ("i(3,5)=3").

Die Zeilen 480-610 geben die ermittelte nächste Generation auf dem Bildschirm aus. Das Array "i(...,...)", das die jeweilige Anzahl an Nachbarn enthält, wird wieder mit Hilfe zweier ineinandergeschachtelter Schleifen Zeile für Zeile und Spalte für Spalte gemäß den geschilderten Regeln abgearbeitet und dabei das Array "z(...,...)", das die Zellenbesetzung enthält, neu aufgebaut.

Immer dann, wenn gilt "h=3", für eine Zelle im Array "i(...,...)" somit drei Nachbarn ermittelt wurden, wird im Array "z(...,...)" der Wert eins eingetragen, da in der betreffenden Zelle ein neues Lebewesen entsteht (Zeile 520). Gleichzeitig wird an der jeweiligen Spielfeldposition das Zeichen "x" ausgegeben.

Hat eine Zelle hingegen weniger als zwei oder mehr als drei Nachbarn, wird in das entsprechende Feld des Arrays "z(...,...)" eine Null eingetragen, ein dort vorhandenes Lebewesen stirbt an Vereinsamung oder Überbevölkerung (Zeile 530). An der entsprechenden Spielfeldposition wird ein Leerzeichen ausgegeben; ein "x", das sich eventuell dort befand, wird dadurch gelöscht, es "stirbt".

Wurde das komplette Spielfeld abgearbeitet, erfolgt ein Sprung zu Zeile 330, die Berechnung der nächsten Generation beginnt.

4.1.4. ASW

ASW ist die gebräuchliche Abkürzung für "Außersinnliche Wahrnehmung". Mit dem vorliegenden Programm können Sie Ihre Fähigkeiten auf diesem Gebiet erproben. Es ist den berühmten Kartentests des amerikanischen Parapsychologen J.B.Rhine nachempfunden. Wenn Sie die mit dem Begriff Parapsychologie verbundenen Begriffe (Außersinnliche Wahrnehmung, Präkognition, Psychokinese) für Realität halten, werden Sie dieses Programm sicher nicht als Spielerei betrachten. Skeptiker können dieses Programm zumindest als amüsantes Gesellschaftsspiel verwenden.

Programmbedienung

Nach dem Programmstart erhalten Sie auf Wunsch eine sehr ausführliche Anleitung, wenn Sie die entsprechende Frage mit "j" beantworten. Anschließend dürfen Sie eingeben, wie viele Durchgänge Sie wünschen. In jedem nun folgenden Durchgang sehen Sie auf dem Bildschirm vier Spielkartensymbole, die mit den Graphikzeichen "Pik", "Herz", "Caro" und "Kreuz" dargestellt werden. Der Computer wählt intern eines dieser vier Symbole aus und Sie sollen raten, um welches es sich handelt.

Drücken Sie als Antwort jene Taste, die dem Anfangsbuchstaben des Symbols entspricht, also "p" für "Pik", "h" für "Herz" und so weiter. Nach jedem erfolgreichen Rateversuch erhalten Sie als "Belohnung" die Meldung "Richtig!!!".

Nach Beendigung des letzten Durchgangs wird Ihnen mitgeteilt, wie groß die Wahrscheinlichkeit (in Prozent) für ein zufällig erreichtes Ergebnis ist. Je kleiner diese Zahl ist, desto geringer ist die Zufallswahrscheinlichkeit und desto größer ist die Wahrscheinlichkeit dafür, daß nicht-zufällige Faktoren - eben ASW - eine Rolle spielten.

```
100 rem ----aussersinnl.wahrnehmung----
110 rem --------(1984/s.baloui)--------
120 :
130 dim a$(19)
140 print chr$(14):rem gross/klein
150 scnclr:print:print:print
160 print tab(4)"Aussersinnliche Wahrnehmung"
170 print tab(4)"---------------------------"
180 print:print:print:print tab(2)"Wuenschen Sie eine An
leitung (j/n) ?"
190 getkey a$
200 if a$="n" then 570
210 :
220 rem ---anleitung---
230 scnclr
240 for a=0 to 18
250 read a$(a)
260 for b=1 to len(a$(a))
270 print mid$(a$(a),b,1);
280 for c=1 to 8
290 nextc,b
300 if a=11 then gosub 970
310 next a
320 gosub 970
330 goto 570
340 :
350 rem ---text---
360 data"Als erstes fragt sie das Programm nach der Anzahl
der von ihnen gewuenschten"
370 data" Durchgaenge (=Versuche).Geben sie die gewuenschte
Zahl ein und druecken sie"
380 data" 'Return'.Sie sehen dann vier Spielkartensymbole.
Versuchen sie das vom"
390 data" Computer ausgewaehlte Symbol zu erraten."
400 data"Positive Versuche werden mit der Meldung"
410 data" 'Richtig' kommentiert.Bei negativen Versuchen er
folgt keine Rueckmeldung."
420 data"Nach der von ihnen gewaehlten Anzahl von Durch
```

```
gaengen werden die Versuche "
430 data"ausgewertet."
440 data"Diese Auswertung kann (je nach Anzahl der Ver
suche) betraechtliche"
450 data" Zeit in Anspruch nehmen.Anschliessend wird ihnen
folgendes mitgeteilt: "
460 data"Die Anzahl der Versuche; die Anzahl der Treffer;
die Wahrscheinlichkeit"
470 data" 'P' mit 'X' Versuchen mindestens 'Y' Treffer zu
erzielen."
480 data"Je geringer die errechnete Wahrscheinlichkeit 'P'
ist,desto "
490 data"unwahrscheinlicher ist es,das ihr Ergebniss nur
ein Zufallsprodukt ist,um so"
500 data" wahrscheinlicher ist es also,das andere Faktoren
eine Rolle spielen "
510 data"(Aussersinnliche Wahrnehmung)."
520 data"Statistisch bedeutsam "
530 data"sind nur Ergebnisse,die unter ca. 5% liegen.Sehr
bedeutsam werden Ergebnisse,"
540 data"wenn sie unter 1% liegen."
550 :
560 rem -----hauptprogramm-----
570 scnclr
580 input"          Wieviele Versuche";b:scnclr
590 for c=1 to b:print:print:print:print
600 print chr$(142):rem gross/graphik
610 print"            AAA  SSS  XXX  ZZZ"
620 print"            ApA  ShS  XkX  ZcZ"
630 print"            AAA  SSS  XXX  ZZZ"
640 a=rnd(0):gosub 760
650 get b$:if b$<>"p" and b$<>"h" and b$<>"c" and b$<>"k"
then 650
660 if b$<>a$ then 700
670 e=e+1
680 print:print:print "               richtig !!!"
690 for d=1 to 500:next d
700 scnclr
710 next c
720 print chr$(14):print "          Bitte warten !!!"
```

```
730 goto 880
740 :
750 rem ---zufallskarte bestimmen---
760 if a<=0.25 then a$="p"
770 if a>0.25 and a<=0.50 then a$="h"
780 if a>0.50 and a<=0.75 then a$="c"
790 if a>0.75 then a$="k"
800 return
810 :
820 rem ---statistische auswertung---
830 fori=1ton
840 x=x+log(i)/log(10)
850 next i
860 m=x-int(x):m1=int(x)
870 return
880 w=0:for h=e to b:p=0:gosub 930:w=w+p:next h
890 scnclr:print:print:print:print
900 print:print tab(5) b"Versuche
910 print:print tab(5) e"Treffer"
920 print:print tab(6)"p="w*100"% !!!":end
930 x=0:n=b:gosub 830:x1=m+m1
940 x=0:n=h:gosub 830:x2=m+m1
950 x=0:n=b-h:gosub 830:x3=m+m1
960 p=10^((x1-(x2+x3))+(-0.60206*h)+(-0.1249387*(b-h))):
return
970 print:print:print " Bitte eine Taste druecken"
980 getkey a$
990 scnclr:return
```

Programmablauf

Der größte Teil des vorliegenden Programms besteht aus Data-Zeilen, in denen die Anleitung enthalten ist. Die Ausgabe dieser Anleitung erfolgt in ungewohnter Form: Auf dem Bildschirm erscheinen nacheinander die einzelnen Zeichen, in gleichmäßigem Tempo.

Dieser Effekt wird dadurch erreicht, daß mit dem Befehl "READ" die Data-Zeilen nacheinander eingelesen werden, um

dann jedoch nicht wie üblich mit einem "PRINT"-Befehl eine Data-Zeile komplett auszugeben, sondern Zeichen für Zeichen der jeweiligen Zeile.

Hierzu wird eine Schleife mit der Schleifenvariablen "b" benutzt, die vom Startwert eins bis zum Endwert "len(a$(a))" läuft. Dieser Endwert ist somit abhängig von der Länge der jeweils auszugebenden Data-Zeile. In dieser Schleife werden die einzelnen Zeichen des Strings "a$(a)" mit dem Befehl "MID$" nacheinander "herausgepickt" und einzeln auf dem Bildschirm ausgegeben (Zeilen 260-290).

Um die Geschwindigkeit der Ausgabe weiter zu verringern, läuft nach jedem ausgegebenen Zeichen eine Verzögerungsschleife mit der Schleifenvariablen "c" vom Startwert eins bis zum Endwert acht. Erst nach Ablauf dieser Schleife wird das nächste Zeichen ausgegeben.

In jedem Durchgang wird zuerst eine Zufallszahl zwischen null und eins erzeugt (Zeile 640). In den Zeilen 760 bis 790 wird dieses Intervall in vier gleiche Teile geteilt. Liegt die Zufallszahl im ersten Viertel (zwischen null und 0.25) wird "a$" der Wert "p" für "Pik" zugeordnet, liegt die Zahl zwischen 0.26 und 0.50, wird "a$" der Wert "h" für "Herz" zugeordnet etc.

Auf diese Weise wird vom PLUS/4 die zu erratende (?) Karte bestimmt. Nun wird auf eine Taste gewartet (Zeile 650). Wenn die gedrückte Taste mit dem gewählten Symbol übereinstimmt, erhält der Benutzer eine entsprechende Meldung ("Richtig!!!"), und die Variable "e", die die Anzahl der Treffer enthält, wird um eins erhöht (Zeilen 670-680).

Wenn der letzte Durchlauf beendet wurde, beginnt die statistische Auswertung. Eine Erläuterung dieses Programmteils will ich jedoch sowohl Ihnen als auch mir ersparen, da dieser nur für einen geringen Leserteil mit größerem Interesse an Statistik interessant sein dürfte.

4.1.5. Indianapolis

Bei "Indianapolis" handelt es sich um ein Autorennen. Dieses Spiel konnte nur mit Hilfe einer in das Programm integrierten Maschinenspracheroutine verwirklicht werden. Das Verständnis des Programmablaufs ist daher nur möglich, wenn Sie über entsprechende Kenntnisse in dieser Programmiersprache verfügen.

Gehen Sie bitte beim Abtippen der im BASIC-Programm integrierten Data-Zeilen möglichst sorgfältig vor. Diese Zeilen enthalten die Maschinenroutine, und ein einziger Tippfehler kann bereits zum Absturz des Programms führen.

Programmbedienung

Die Spielidee besteht darin, dem Verlauf einer Rennstrecke, die auf den Bildschirm gemalt wird, möglichst lange zu folgen, bevor der Rennwagen mit einer der seitlichen Streckenbegrenzungen kollidiert. Maximal zehn Fahrzeuge stehen zur Verfügung. Jede Kollision kostet Sie eines dieser Fahrzeuge. Je länger Sie den Rennwagen auf der Strecke halten können, desto mehr Kilometer legen Sie zurück.

Nach dem Starten des Programms können Sie eine beliebige Geschwindigkeit im Bereich eins bis neun anwählen. Je höher die eingegebene Zahl, desto höher wird das Tempo Ihres Rennwagens sein.

Auf dem Bildschirm erscheint nun die Rennstrecke und - zwischen der rechten beziehungsweise linken Streckenbegrenzung - Ihr Rennwagen. Ihre Aufgabe besteht darin, den Wagen mit Hilfe der beiden Tasten "Cursor rechts" und "Cursor links" auf der Strecke zu halten.

Je höher die gewählte Geschwindigkeit ist, desto schwieriger wird es für Sie, dem Streckenverlauf zu folgen, desto mehr Kilometer legen Sie jedoch in einer bestimmten Zeiteinheit zurück.

Jede Kollision vermindert Ihren "Fuhrpark" um einen Rennwagen. Das Spiel ist zu Ende, wenn alle zehn Fahrzeuge am Streckenrand zerschellt sind.

Programmlisting

```
100 rem -----autorennen-----
110 rem ---(s.baloui/1986)--
120 :
130 rem ---variablenliste---
140 rem t=linker streckenrand (spalte)
150 rem p1=bisherige pos.auto (spalte)
160 rem p2=neue pos.auto (spalte)
170 rem zk=zurueckgelegte kilometer
180 rem zo=zeit bei start
190 :
200 rem ---variablen---
210 cr$=chr$(29):rem 'cursor rechts'
220 cl$=chr$(157):rem 'cursor links'
230 :
240 rem ---datas einlesen---
250 sa=1630:rem startadresse
260 read a:if a<>-1 then poke sa,a:sa=sa+1:goto 260
270 :
280 color 0,1:color 4,1:color 1,2
290 scnclr
300 print tab(5)"geschwindigkeit (1-9) ?"
310 getkey a$:g=val(a$):if g<1 then 310
320 :
330 rem -----hauptprogramm-----
340 for i=1 to 10
350 zo=ti:rem zeit retten
360 scnclr
370 print tab(5)"zurueckgelegte kilometer:"int(zk)
380 print tab(5)"verbleibende fahrzeuge"11-i
390 print:print tab(5)"start mit beliebiger taste"
400 vol 8:rem volle lautstaerke
410 poke 239,0:getkey a$:if a$=cr$ or a$=cl$ then 410:rem
cursortaste?
```

```
420 for ii=1 to 500:next:rem warteschleife
430 :
440 rem ---strecke/auto bewegen---
450 t=15:p1=20:p2=20:rem strecken- und autoposition
initialisieren
460 poke 216,t:poke 217,p1:poke 218,p2:rem positionen ueber
geben
470 char 1,0,24,""
480 sys 1630:rem maschinenrout.aufrufen
490 sound 3,200,19-2*g:rem geraeuschkulisse
500 get a$:rem cursortaste betaetigt?
510 p1=p2:rem alte autopos.retten
520 if a$=cr$ then if p2<37 then p2=p2+1:goto 540
530 if a$=cl$ then if p2 then p2=p2-1
540 poke 217,p1:poke 218,p2:rem alte/neue autopos.
uebergeben
550 if x<>2 then x=x+1:goto 570:else x=0
560 a=int(rnd(0)*2):poke 219,a:rem zufalsszahl fuer
streckenpos.
570 t=peek(216):rem streckenpos.holen
580 if t<p1 and t+14>p1+2 then 480:rem kollision?
590 zk=zk+(ti-zo)/300*g:rem zurueckgle.kilometer
600 vol 0:rem lautstaerke null
610 next
620 :
630 rem ---durchgaenge beendet---
640 char 1,0,0," summe zurueckgelegter kilometer:"
+str$(int(zk))
650 char 1,10,3,"noch eine runde (j/n) ?"
660 getkey a$:if a$<>"j" and a$<>"n" then 660
670 if a$="j" then run:else end
680 :
690 rem ---datas---
700 data 165,219,74,144,9,165,216,240,11,198,216,76,116,6,
165,216,201,25,176
710 data 245,230,216,169,32,160,3,166,217,157,112,15,157,
152,15,157,192,15
720 data 232,136,208,243,169,13,32,210,255,169,81,166,218,
157,112,15,157,114
```

```
730 data 15,157,192,15,157,194,15,169,160,157,113,15,157,
153,15,157,193,15
740 data 166,216,169,118,157,192,15,169,117,157,206,15,
96,32,32,-1
```

Programmablauf

Da dieses Programm eine der wenigen in diesem Buch verwendeten Maschinenroutinen benötigt, empfehle ich Ihnen, diese Erläuterung zu übergehen, sollten Sie keine entsprechenden Kenntnisse besitzen.

Die Assemblerroutine hat die Aufgabe, die Strecke (zwei Graphikzeichen) auf dem Bildschirm auszugeben, und das Fahrzeug - entsprechend den betätigten Cursortasten - in der untersten Bildschirmzeile zu bewegen.

Das BASIC-Programm erledigt die mehr sekundären Aufgaben wie zum Beispiel die Bildschirmausgabe der verbleibenden Fahrzeuge, der zurückgelegten Kilometer (Zeile 370-390) und vor allem die Übergabe verschiedener Parameter an das Maschinenprogramm.

Diesem wird die Spalte mitgeteilt, an der der linke Streckenrand beginnt (der rechte Streckenrand befindet sich immer an dieser Spaltenposition plus 13 Spalten, die Streckenbreite beträgt somit zwölf Spalten), und sowohl die bisherige Fahrzeugposition als auch die aktuelle und eventuell - Cursortasten! - gegenüber der bisherigen leicht veränderte Position.

Die benötigten Parameter und die Speicherstellen, in denen sie übergeben werden:

$D8=216:	Linker Streckenrand
$D9=217:	Alte Fahrzeugspalte
$DA=218:	Neue Fahrzeugspalte

Der Ablauf der Maschinenroutine (Listing am Kapitelende): Zuerst wird die Position der Streckenbegrenzungen ermittelt. Die betreffende - zufällig ermittelte - Spalte (Position der linken Begrenzung) wird zwar vom BASIC-Programm übergeben. Um zu verhindern, daß die Rennstrecke sich rechts oder links aus dem Bildschirm herausbewegt, wird die übergebene Spalte eventuell korrigiert.

Wenn sich der linke Streckenrand am linken Bildschirmrand befindet, wird er um eine Spalte nach rechts versetzt. Befindet sich hingegen der rechte Streckenrand an der rechten Bildschirmbegrenzung, wird er um eine Spalte nach links versetzt.

Das Fahrzeug wird anschließend an seiner bisherigen Position gelöscht, das heißt mit Leerzeichen überschrieben. Der folgende Zeilenvorschub (Ausgabe des Zeichens "Return") führt zur Bewegung der Rennstrecke, da ein Zeilenvorschub in der untersten Bildschirmzeile ein "Hochscrollen" des gesamten Bildschirminhaltes bewirkt, ein Verschieben aller Zeilen nach oben. Auf diesem Trick beruht die gesamte Bewegung der Rennstrecke.

An der in $DA übergebenen neuen Position wird das Fahrzeug auf dem Bildschirm ausgegeben, bevor in der untersten Bildschirmzeile die beiden Graphikzeichen für die Streckenbegrenzung gezeichnet werden.

Wie Sie sehen, bewirkt die Maschinenroutine alle für das Spiel notwendigen Bewegungsabläufe, die Bewegung der Rennstrecke und die Steuerung des Rennwagens.

Das BASIC-Programm übernimmt die weniger zeitkritischen Aufgaben wie zum Beispiel die Tastaturabfrage, die entsprechende Änderung der Variablen "p1" und "p2", die die alte und neue Position des Rennwagens beinhalten, und die Übergabe dieser Parameter an die Maschinenroutine (Zeile 500-540).

Weiterhin ermittelt das BASIC-Programm bei jedem zweiten Durchgang eine Zufallszahl, die angibt, ob sich die Rennstrecke um eine Spalte nach rechts oder nach links bewegen soll, und

übergibt diese Zufallszahl ebenfalls an die Maschinenroutine (Zeile 550-560).

Die letzte wichtige Funktion des BASIC-Programms besteht in der Kollisionsabfrage, dem Vergleich der aktuellen Fahrzeugposition mit der momentanen Position der Streckenbegrenzungen, das heißt des linken und des rechten Randes der Rennstrecke (Zeile 570-580).

Die Position des linken Randes - die eventuell durch die Maschinenroutine korrigiert wurde - ergibt sich durch Abfrage der Speicherstelle 216 (Zeile 570). Der rechte Streckenrand befindet sich 13 Spalten rechts vom linken Rand entfernt. Eine Kollision erfolgte genau dann, wenn sich das Fahrzeug nicht mehr innerhalb der durch diese Grenzen gegebenen Strecke befindet (Zeile 580).

Neue Streckenposition ermitteln:

```
. 0663  a5 d8     lda $d8
. 0665  f0 0b     beq $0672
. 0667  c6 d8     dec $d8
. 0669  4c 74 06  jmp $0674
. 066c  a5 d8     lda $d8
. 066e  c9 19     cmp #$19
. 0670  b0 f5     bcs $0667
. 0672  e6 d8     inc $d8
```

Fahrzeug an alter Position löschen:

```
. 0674  a9 20     lda #$20
. 0676  a0 03     ldy #$03
. 0678  a6 d9     ldx $d9
. 067a  9d 70 0f  sta $0f70,x
. 067d  9d 98 0f  sta $0f98,x
. 0680  9d c0 0f  sta $0fc0,x
```

```
. 0683  e8        inx
. 0684  88        dey
. 0685  d0 f3     bne $067a
```

Zeilenvorschub:

```
. 0687  a9 0d     lda #$0d
. 0689  20 d2 ff  jsr $ffd2
```

Fahrzeug an neuer Position ausgeben:

```
. 068c  a9 51     lda #$51
. 068e  a6 da     ldx $da
. 0690  9d 70 0f  sta $0f70,x
. 0693  9d 72 0f  sta $0f72,x
. 0696  9d c0 0f  sta $0fc0,x
. 0699  9d c2 0f  sta $0fc2,x
. 069c  a9 a0     lda #$a0
. 069e  9d 71 0f  sta $0f71,x
. 06a1  9d 99 0f  sta $0f99,x
. 06a4  9d c1 0f  sta $0fc1,x
```

Streckenbegrenzung ausgeben:

```
. 06a7  a6 d8     ldx $d8
. 06a9  a9 76     lda #$76
. 06ab  9d c0 0f  sta $0fc0,x
. 06ae  a9 75     lda #$75
. 06b0  9d ce 0f  sta $0fce,x
. 06b3  60        rts
```

4.1.6. Space Invaders

Das vorliegende Programm ist eine der unzähligen Variationen der sogenannten "Space Invaders"-Idee und stellt dem Spieler die

Aufgabe, den aus dem All kommenden Angreifer abzuwehren, der die Erde bedroht.

Die Steuerung dieses Angreifers übernimmt eine Maschinenroutine, deren Verständnis entsprechende Maschinensprachekenntnisse voraussetzt.

Programmbedienung

Wählen Sie bitte nach dem Programmstart eine der neun möglichen Geschwindgkeitsstufen (eins bis neun). Nun folgen zehn Durchgänge, in denen jeweils der Angreifer in der obersten Bildschirmzeile erscheint und sich quer über den den ganzen Bildschirm nach unten bewegt. Ihre "Festung" befindet sich in der untersten Bildschirmzeile.

Zur Abwehr des Angreifers steht Ihnen ein Geschoß zur Verfügung, das Sie mit einer beliebigen Taste abfeuern können. Wenn Sie den Angreifer nicht treffen, bevor er Ihre Basis zerstört (das heißt, bevor er den unteren Bildschirmrand erreicht), werden Ihnen Punkte abgezogen (sofern Ihr Punktestand nicht gleich null ist!).

Wenn Sie den Angreifer zuvor treffen, erhalten Sie umso mehr Punkte, je höher die gewählte Geschwindigkeit ist und umso früher es Ihnen gelingt, den Angreifer abzuwehren. Zur Erzielung eines möglichst hohen Punktestandes ist es daher notwendig, den Angreifer möglichst bereits dann zu treffen, wenn er sich noch in den oberen Bildschirmzeilen befindet.

Nach zehn Durchgängen wird Ihnen die erzielte Punktzahl gemeldet, und Sie werden gefragt, ob Sie einen weiteren Angriff abwehren sollen ("weiter (j/n) ?"). Mit "j" starten Sie eine weitere Runde, mit "n" wird das Programm beendet.

Programmlisting

```
100 rem ----- invaders -----
110 rem --s.baloui/jan 86---
120 :
130 rem ---variablenliste---
140 rem gf=geschossflag (1=geschoss abgefuert, sonst 0)
150 rem g=geschwindigkeit
160 rem tf=trefferflag (0=kein;1=rampe;sonst=treffer)
170 rem pt=punktezahl
180 rem gz=geschosszeile
190 :
200 rem ---variablen---
210 ge$=chr$(94):rem geschosszeichen
220 ra$=chr$(166):rem graphikzeichen fuer geschossrampe
230 :
240 rem ---datas einlesen---
250 sa=1630:rem startadresse
260 read a:if a<>-1 then poke sa,a:sa=sa+1:goto 260
270 :
280 vol 8:rem volle lautstaerke
290 scnclr
300 print tab(13)"space invaders"
310 print tab(13)"--------------"
320 print:print:print
330 print tab(8)"geschwindigkeit (1-9) ?"
340 getkey a$:if a$<"1" and a$>"9" then 340
350 g=10-val(a$)
360 poke 217,g:rem tempo uebergeben
370 poke 216,30:rem zeichencode uebergeben
380 color 0,1:color 4,1:color 1,2
390 scnclr
400 :
410 rem -----hauptprogramm-----
420 for i=1 to 10
430 gf=0:rem geschossflag initialisieren
440 scnclr
450 char 1,0,24,"treffer:"+str$(tr)
460 char 1,28,24,"pkt.:"+str$(pt)+"  "
470 char 1,19,24,ra$+ra$+ra$:rem geschossrampe zeichnen
```

```
480 sys 1630:rem routine starten
490 :
500 if gf then gosub 620:else geta$:if a$<>"" and gf=0 then
gf=1:gz=23
510 tf=peek(218):if tf=0 then 500:rem trefferflag
520 if tf<>1 then gosub 680:goto 560:rem treffer?
530 char 1,3,5,"feind kam durch!    punktabzug!!!"
540 pt=pt-100:if pt<0 then pt=0:rem punktabzug
550 for ii=1 to 2000:next:rem warteschleife
560 next
570 :
580 char 1,14,5,"weiter (j/n) ?"
590 getkey a$:if a$="j" then run:else end
600 :
610 rem ---geschoss---
620 char 0,20,gz," "
630 gz=gz-1
640 if gz<>-1 then char 0,20,gz,ge$:else gf=0
650 return
660 :
670 rem ---treffer---
680 char 1,20,gz,ge$
690 for ii=800 to 500 step -10:sound 3,i,1:next
700 tr=tr+1:pt=pt+(10-g)+23-gz
710 char 1,8,24,str$(tr)+" "
720 char 1,33,24,str$(pt)+" "
730 char 1,17,24,"treffer!"
740 for ii=1 to 2000:next
750 return
760 :
770 rem ---datas---
780 data120,169,123,162,6,141,20,3,142,21,3,169,255,162,
11,133,220,134,221
790 data165,217,133,219,169,0,133,218,88,96,198,219,208,
47,165,217,133,219
800 data160,0,177,220,170,169,32,145,220,228,216,240,22,
230,220,208,2,230
810 data221,165,221,201,16,208,4,230,218,208,17,177,220,
197,216,208,4,133
```

```
820 data218,240,7,169,42,145,220,76,14,206,169,14,162,206,
141,20,3,142,21
830 data3,208,241,-1
```

Programmablauf

Um das Spiel durch eine möglichst hohe Geschwindigkeit des Angreifers interessant zu gestalten, wurde der entsprechende Programmteil in Maschinensprache geschrieben, und zwar als sogenannte "IRQ-Routine", die in den Systeminterrupt des PLUS/4 eingebunden ist.

Der Systeminterrupt und die Einbindung eigener IRQ-Routinen wird eingehend im Kapitel "Laufschrift" erläutert. Ich beschränke mich in diesem Kapitel daher auf den eigentlichen Funktionsablauf der Maschinenroutine.

Diese Routine hat die Aufgabe, den - durch ein Graphikzeichen dargestellten - Angreifer Spalte für Spalte über den Bildschirm zu bewegen, ausgehend vom oberen linken Bildschirmrand. Diese Bewegung soll enden, wenn entweder der rechte untere Bildschirmrand erreicht und damit die Festung des Verteidigers zerstört wurde, oder aber, wenn sich an der aktuellen Position des Angreifers das Geschoß - ein weiteres Graphikzeichen - befindet, das dem Verteidiger zur Abwehr zur Verfügung steht. In letzterem liegt ein Treffer vor, der dem BASIC-Programm signalisiert werden muß.

Nach der Initialisierung der Routine befindet sich in $DC/$DD ein Zeiger, der auf den Anfang des Bildschirmzeigers weist. Bei jedem Aufruf der Routine wird der Zeiger inkrementiert und weist damit auf die nächste Bildschirmposition, entsprechend der Weiterbewegung des Angreifers.

Der Ablauf der Routine: Der Angreifer wird an der momentanen Position gelöscht, das heißt diese Position wird mit einem Leerzeichen überschrieben. Danach wird abgefragt, ob sich an

dieser Position das vom BASIC-Programm (Zeile 370) in Speicherstelle $D8 (= dezimal 216) übergebene Graphikzeichen befand, welches das Geschoß symbolisiert.

Wenn ja, erhält die Speicherstelle $DA (= dezimal 218) den Wert 32. Der Inhalt dieser Speicherstelle wird vom BASIC-Programm abgefragt, das an dem Wert 32 erkennt, daß ein Treffer vorliegt (Zeile 510-520).

Liegt kein Treffer vor, wird wie beschrieben der Zeiger $DC/$DD erhöht, der auf die aktuelle Bildschirmposition weist. Liegt diese aktuelle Position anschließend außerhalb des Bildschirmspeichers, wurde der Angreifer bereits über den gesamten Bildschirm bewegt und die Festung des Verteidigers zerstört. In diesem Fall erhält die Speicherstelle $D8 den Wert eins. An diesem Wert erkennt das BASIC-Programm, das die Verteidigung mißlang (Zeile 530-560) und zieht zur "Strafe" 100 Punkte vom bisher erreichten Punktestand ab.

Wenn weder ein Treffer erfolgte noch die Verteidigerfestung zerstört wurde, wird abgefragt, ob sich an der neuen Bildschirmposition ein Geschoß befindet, an der der Angreifer nun gezeichnet werden sollte. Wenn ja, wird dem BASIC-Programm ebenfalls ein Treffer signalisiert. Ist dies nicht der Fall, wird der Angreifer, das heißt das Graphikzeichen mit dem Code $2A (= dezimal 42) an der neuen Position auf dem Bildschirm ausgegeben und wurde damit genau eine Spalte weiterbewegt.

Eine IRQ-Routine läuft völlig unabhängig vom BASIC-Programm. Den letzten Programmteil bildet daher das Ausschalten der Routine. Dieses Ausschalten wird vorgenommen, wenn ein Treffer erfolgte oder die Verteidigerfestung zerstört wurde.

Dank der IRQ-Routine, die den eigentlichen Programmkern bildet, hat das BASIC-Programm nur wenige Aufgaben, die vorwiegend in der Ausgabe verschiedener Meldungen bestehen (Punktezahl, Treffermeldung etc.). Die wesentlichste Aufgabe besteht in der Geschoßbewegung und der Trefferausgabe (Zeile 610-750).

Das Geschoß wird bewegt, indem es an der alten Position "gz" gelöscht wird, "gz" erniedrigt und das Geschoß anschließend an der neuen Position ausgegeben wird (Zeile 610-650).

Die Trefferausgabe besteht fast ausschließlich aus "CHAR"-Befehlen, mit denen der Treffer gemeldet und die neue Punktzahl mitgeteilt wird (Zeile 670-750).

IRQ-Routine initialisieren:

```
. 065e  78        sei
. 065f  a9 7b     lda #$7b
. 0661  a2 06     ldx #$06
. 0663  8d 14 03  sta $0314
. 0666  8e 15 03  stx $0315
. 0669  a9 ff     lda #$ff
. 066b  a2 0b     ldx #$0b
. 066d  85 dc     sta $dc
. 066f  86 dd     stx $dd
. 0671  a5 d9     lda $d9
. 0673  85 db     sta $db
. 0675  a9 00     lda #$00
. 0677  85 da     sta $da
. 0679  58        cli
. 067a  60        rts
```

IRQ-Routine:

```
. 067b  c6 db     dec $db
. 067d  d0 2f     bne $06ae
. 067f  a5 d9     lda $d9
. 0681  85 db     sta $db
. 0683  a0 00     ldy #$00
. 0685  b1 dc     lda ($dc),y
. 0687  aa        tax
. 0688  a9 20     lda #$20
. 068a  91 dc     sta ($dc),y
. 068c  e4 d8     cpx $d8
```

```
. 068e  f0 16     beq $06a6
. 0690  e6 dc     inc $dc
. 0692  d0 02     bne $0696
. 0694  e6 dd     inc $dd
. 0696  a5 dd     lda $dd
. 0698  c9 10     cmp #$10
. 069a  d0 04     bne $06a0
. 069c  e6 da     inc $da
. 069e  d0 11     bne $06b1
. 06a0  b1 dc     lda ($dc),y
. 06a2  c5 d8     cmp $d8
. 06a4  d0 04     bne $06aa
. 06a6  85 da     sta $da
. 06a8  f0 07     beq $06b1
. 06aa  a9 2a     lda #$2a
. 06ac  91 dc     sta ($dc),y
. 06ae  4c 0e ce  jmp $ce0e
```

IRQ-Routine ausschalten:

```
. 06b1  a9 0e     lda #$0e
. 06b3  a2 ce     ldx #$ce
. 06b5  8d 14 03  sta $0314
. 06b8  8e 15 03  stx $0315
. 06bb  d0 f1     bne $06ae
```

4.2. Graphik

Zwei der folgenden Programme aus dem Bereich Graphik, die Programme "Malprogramm" und "Funktionsdarstellung", verwenden die hochauflösende Graphik des PLUS/4. Obwohl in beiden Programmen in dieser "Hi-Res"-Graphik gearbeitet wird, soll Sie das nicht daran hindern, in eigenen Programmen auch den sogenannten "Multicolormodus" des PLUS/4 zu verwenden. Die Auflösung in diesem Modus ist zwar geringer als die Auflösung des "Hi-Res-Modus", dafür können jedoch mehrere Farben gleichzeitig auf dem Bildschirm dargestellt werden.

4.2.1. Malprogramm

Das vorliegende Programm ist ein Mal- oder Zeichenprogramm, das es Ihnen erlaubt, Zeichnungen beliebiger Art anzufertigen. Folgende Möglichkeiten stehen Ihnen zur Verfügung: Punkte setzen und löschen, Linien ziehen und löschen, Rechtecke malen und löschen, Flächen ausfüllen oder löschen.

Programmbedienung

Zur Orientierung sehen Sie an der momentanen Malposition immer ein Fadenkreuz. Dieses Fadenkreuz können Sie mit den Cursortasten beliebig bewegen. Das Programm unterscheidet drei verschiedene Modi:

Bewegungsmodus:

In diesem Modus können Sie das Fadenkreuz bewegen, ohne eventuell bereits vorhandene Zeichnungen zu beeinflußen. Wenn Sie zum Beispiel mit dem Fadenkreuz eine Linie überqueren, ist diese nach der Überquerung wieder vollständig vorhanden, ohne daß der Kreuzungspunkt zerstört wird.

In diesem Modus kann nicht gemalt werden, weder Punkte noch Linien oder Rechtecke können gezeichnet werden. Auch das Ausfüllen von Flächen ist nicht möglich. In diesem Modus kann nur das Fadenkreuz bewegt werden.

Malmodus:

Wenn Sie das Fadenkreuz im "Mal-Modus" bewegen, hinterläßt es eine "Spur", es werden Punkte gezeichnet. Das Fadenkreuz fungiert als Zeichenstift. Dieser Modus muß ebenfalls angewählt werden, wenn Sie Linien oder Rechtecke zeichen, oder aber eine Fläche ausfüllen wollen.

Löschmodus:

Das Fadenkreuz wirkt in diesem Modus wie ein Radiergummi, das alles auslöscht, was ihm auf seinem Weg begegnet. Wird in diesem Modus die Funktion "Linie zeichnen" gewählt, löscht das Radiergummi die zwischen den Endpunkten der Linie vorhandenen Punkte. Gleiches gilt für das Zeichnen von Rechtecken, die nun gelöscht werden. Das Ausfüllen von Flächen führt dazu, daß die komplette Fläche "ausradiert" wird.

In welchem Modus Sie sich gerade befinden, wird ständig in der untersten Bildschirmzeile angezeigt. Zum Umschalten zwischen den verschiedenen Kommandos dient eines der im folgenden besprochenen Kommandos, die mit Hilfe der Funktionstasten angewählt werden.

Moduswechsel (F1):

Nach dem Starten des Programms befinden Sie sich im "Bewegungs-Modus". Wenn Sie "F1" drücken, gelangen Sie in den "Lösch-Modus". Ein weiterer Druck auf "F1" führt zum "Mal-Modus", und die nächste Betätigung wieder zum Modus "Bewegen". "F1" wirkt wie ein Schalter: Jede Betätigung schaltet in den nächsten der drei Modi um.

Parameterpunkt setzen (F2):

Für die folgenden Funktionen "Linie ziehen/löschen" und "Rechteck zeichnen/löschen" wird jeweils die Angabe zweier Punkte benötigt. Einer der beiden Punkte wird immer durch die momentane Position des Fadenkreuzes angezeigt. Der zweite Punkt muß von Ihnen angegeben werden, indem Sie an der jeweiligen Position die Taste "F2" drücken. An dieser Position wird ein Punkt gezeichnet, der die eine Koordinate der zu zeichnenden Linie beziehungsweise des Rechtecks angibt.

Parameterpunkt löschen (F5):

Sollten Sie bereits den Parameterpunkt gesetzt haben und sich im nachhinein eine andere Position für die zu zeichnende Linie beziehungsweise das Rechteck überlegen, löschen Sie bitte diesen Punkt, indem Sie "F5" betätigen. Der markierte Parameterpunkt wird gelöscht werden, unabhängig davon, an welcher Position sich momentan das Fadenkreuz befindet.

Linie ziehen/löschen (F3):

Um eine Linie zu ziehen, werden zwei Parameter benötigt, die beiden Endpunkte der Linie. Markieren Sie einen der Endpunkte, indem Sie an der gewünschten Position "F2" betätigen. Schalten Sie nun - sofern er noch nicht eingeschaltet ist - in den "Bewegungs-Modus" und bewegen Sie das Fadenkreuz zum anderen gewünschten Endpunkt der Linie. Schalten Sie in den "Mal-Modus" und betätigen Sie die Taste "F3". Zwischen den beiden gewählten Endpunkten wird eine Linie gezeichnet.

Um eine vorhandene Linie ganz oder teilweise zu löschen, markieren Sie bitte einen der Endpunkte, schalten Sie in den "Bewegungs-Modus", führen Sie das Fadenkreuz zum anderen Endpunkt, schalten Sie anschließend den "Lösch-Modus" ein und betätigen Sie "F3". Im "Lösch-Modus" werden alle Punkte zwischen den Endpunkten einer Linie nicht gezeichnet, sondern gelöscht.

Rechteck zeichnen/löschen (F6):

Markieren Sie bitte wie beschrieben einen Punkt mit "F2". Dieser Punkt stellt die obere linke Ecke des zu zeichnenden Rechtecks dar. Bewegen Sie das Fadenkreuz - im "Bewegungs-Modus" - zu jener Position, die die rechte untere Ecke des Rechtecks ergeben soll, und drücken Sie - nachdem Sie in den "Mal-Modus" geschaltet haben - die Taste "F6". Das Rechteck zwischen den angegebenen Eckkoordinaten wird nun gezeichnet.

Wie Sie sich bereits denken können, werden alle Punkte, die sich auf den Ecklinien des Rechtecks befinden, gelöscht statt gezeichnet, wenn Sie den gleichen Vorgang noch einmal durchführen, jedoch den "Lösch-Modus" einschalten, bevor Sie die Taste "F6" betätigen.

Fläche ausfüllen/löschen (F8/Help):

Mit der Taste "F8" oder auch "Help" kann eine beliebige umgrenzte Fläche ausgefüllt oder gelöscht werden, je nachdem in welchem Modus Sie sich befinden, wenn Sie diese Taste betätigen.

Zeichnen Sie bitte auf die beschriebene Weise ein Rechteck, schalten Sie in den "Bewegungs-Modus" und führen Sie das Fadenkreuz in das Rechteck hinein. Das Fadenkreuz ist nun von einer umgrenzten Fläche umgeben, dem gerade gezeichneten Rechteck.

Schalten Sie nun bitte auf den "Mal-Modus" um und drücken Sie "F8" (Help). Das Rechteck wird vollständig ausgefüllt werden. Wenn Sie den "Lösch-Modus" einschalten und anschließend "F8" betätigen, wird das nun ausgefüllte Rechteck gelöscht werden.

Seien Sie mit diesem Befehl bitte äußerst vorsichtig! Wenn die Fläche, innerhalb derer sich der Cursor befindet, nicht völlig geschlossen ist, wird der komplette Bildschirm ausgefüllt beziehungsweise gelöscht!

Graphik ausschalten (Esc):

Durch Betätigung der Taste "Esc" können Sie Ihre "Malerei" jederzeit abbrechen und die hochauflösende Graphik abschalten.

Zusammenfassung:

Es stehen Ihnen verschiedene Funktionen und Modi zur Verfügung. Die Funktionen haben unterschiedliche Auswirkungen, je nachdem in welchem Modus Sie sich befinden.

Im "Bewegungs-Modus" kann nur das Fadenkreuz bewegt werden (ohne Auswirkungen auf das Bild). Im "Mal-Modus" werden Punkte gesetzt, Linien oder Rechtecke gezogen und Flächen ausgefüllt. Im "Lösch-Modus" wirken die Funktionen umgekehrt: Punkte, Linien etc. werden gelöscht. Die einzelnen Funktionen:

F1: Modus umschalten
F2: Parameterpunkt setzen
F5: Parameterpunkt löschen
F3: Linie ziehen/löschen
F6: Rechteck zeichnen/löschen
F8 (HELP): Fläche ausfällen/löschen
Esc: Graphik ausschalten

Programmlisting

```
100 rem -----malprogramm-----
110 rem -(jan 1986/s.baloui)-
120 :
130 rem ---kommandostring erzeugen---
140 for i=1 to 8:key i,chr$(132+i):k$=k$+chr$(132+i):next
150 k$=chr$(29)+chr$(157)+chr$(17)+chr$(145)+k$
160 :
170 rem ---initialisierung---
180 crs$=chr$(160)+chr$(64)+chr$(160)+chr$(2)+chr$(0)
+chr$(2)+chr$(0)
190 graphic 1,1
200 x=160:y=100:rem startposition
210 sshape old$,x-1,y-1,x+1,y+1
220 gshape crs$,x-1,y-1
```

```
230 locate x,y
240 mode=1:gosub 640
250 :
260 rem -----hauptprogramm-----
270 getkey a$
280 if a$=chr$(27) then graphic0:end
290 k=instr(k$,a$)
300 gshape old$,x-1,y-1
310 locate x,y
320 on k gosub 400,460,520,580,640,730,840,430,780,890,
430,940
330 x=rdot(0):y=rdot(1)
340 sshape old$,x-1,y-1,x+1,y+1
350 gshape crs$,x-1,y-1
360 locate x,y
370 goto 270
380 :
390 rem ---cursor rechts---
400 if x>317 then 420
410 if mode<>2 then draw mode to x+2,y
420 locate x+2,y
430 return
440 :
450 rem ---cursor links---
460 if x<2 then 490
470 if mode<>2 then draw mode to x-2,y
480 locate x-2,y
490 return
500 :
510 rem ---cursor unten---
520 if y>197 then 550
530 if mode<>2 then draw mode to x,y+2
540 locate x,y+2
550 return
560 :
570 rem ---cursor oben---
580 if y<2 then 610
590 if mode<>2 then draw mode to x,y-2
600 locate x,y-2
610 return
```

```
620 :
630 rem ---modus umschalten---
640 mode=mode+1:if mode>2 then mode=0
650 on mode+1 gosub 680,690,700
660 char 1,0,24,"modus: "+mode$
670 return
680 mode$="loeschen":return
690 mode$="malen   ":return
700 mode$="bewegen ":return
710 :
720 rem ---parameterpunkt setzen---
730 xp=x:yp=y
740 draw 1,xp,yp
750 return
760 :
770 rem ---parameterpunkt loeschen---
780 draw 0,xp,yp
790 xp=-1
800 locate x,y
810 return
820 :
830 rem ---linie ziehen---
840 if xp<>-1 then if mode<>2 then draw mode to xp,yp
850 locate x,y
860 return
870 :
880 rem ---rechteck malen---
890 if xp<>-1 then if mode<>2 then box mode,x,y,xp,yp
900 locate x,y
910 return
920 :
930 rem ---flaeche---
940 if mode<>2 then paint mode
950 return
```

Programmablauf

Vor der eigentlichen Erläuterung ein Hinweis: Wenn Sie die Graphikbefehle im Handbuch zum PLUS/4 nachlesen, werden

Sie feststellen, daß bei den Befehlen "DRAW", "LINE" und "PAINT" ganz oder teilweise die Angabe von Koordinaten entfallen kann. In diesem Fall verwendet der BASIC-Interpreter als Koordinaten die aktuelle Position des Graphikcursors. Merken Sie sich diese Besonderheit bitte, da im vorliegenden Programm reichlich Gebrauch von dieser Möglichkeit gemacht wird.

In den Zeilen 130-150 wird ein String "k$" gebildet, der die Cursorsteuerzeichen und alle den Funktionstasten neu zugeordnete Zeichen (chr$(133)-chr$(140)) enthält.

Der String "crs$" (Zeile 180) enthält das Fadenkreuz, den "Graphikcursor", und wurde mit dem Befehl "SSHAPE" gebildet (jedoch nicht im Rahmen dieses Programms!).

In Zeile 190 wird die hochauflösende Graphik eingeschaltet, anschließend wird die Ausgangsposition (Bildschirmmitte) in den Variablen "x" und "y" festgelegt.

Um die folgenden Befehle zu verstehen, müssen Sie wissen, daß das Fadenkreuz sowohl in X- als auch in Y-Richtung eine Ausdehnung von jeweils drei Punkten besitzt.

Fadenkreuz:

```
x x
 x
x x
```

Bevor dieses Fadenkreuz an der Ausgangsposition, der Koordinate 160/100 gezeichnet wird, wird der "Untergrund" unter dem zu malenden Fadenkreuz gerettet, der momentan aus neun (drei mal drei) nicht gesetzten Punkten besteht. Anschließend wird das Fadenkreuz gezeichnet (Zeile 220).

Der Graphikcursor wird auf die Ausgangsposition gesetzt (Zeile 230) und danach der Modus initialisiert. Zum Modus ist zu

sagen, daß die Variable "mode" drei verschiedene Werte annehmen kann.

1. mode=0: Lösch-Modus
2. mode=1: Mal-Modus
3. mode=2: Bewegungs-Modus

"mode" erhält in Zeile 240 den Wert eins, da direkt im Anschluß daran das Unterprogramm "Modus-Wechsel" aufgerufen wird, das in den nächsten Modus schaltet und den aktuellen Modus auf dem Bildschirm anzeigt. Nach Aufruf dieses Unterprogramms besitzt "mode" den gewünschten Wert zwei, der "Bewegungs-Modus" ist eingeschaltet.

Das eigentliche Hauptprogramm umfaßt die Zeilen 260-370. Seine Aufgabe besteht darin, auf ein Kommando zu warten und das zugehörige Unterprogramm aufzurufen.

Durch verschiedene Kommandos wird das Fadenkreuz bewegt (Cursortasten). Der Untergrund, über den es sich bewegt, wird dabei überschrieben, wenn dieses Problem nicht speziell berücksichtigt wird.

Daher wurde bei der Initialisierung dieser Untergrund vor der Ausgabe des Fadenkreuzes in der Variablen "old$" gespeichert ((siehe "SSHAPE/GSHAPE" im Handbuch). Vor der Ausführung eines Kommandos wird der in "old$" gespeicherte Untergrund ausgegeben und das Fadenkreuz damit gelöscht. Anstelle des Fadenkreuzes werden wieder die zuvor an dieser Position gesetzten Punkte sichtbar (Zeile 300).

Da die Befehle "GSHAPE" und "SSHAPE" die Position des Graphikcursors verändern, wird dieser erneut auf die in "x" und "y" enthaltene Position gesetzt (Zeile 310).

Nachdem das betreffende Unterprogramm ausgeführt wurde (Zeile 320), wird die aktuelle Position des Graphikcursors - die eventuell geändert wurde (Cursortasten) - in "x" und "y" festgehalten (Zeile 330).

Bevor nun das Fadenkreuz an diese neue Position gesetzt wird (Zeile 350), wird der momentane Untergrund in die Variable "old$" übertragen (Zeile 340). Aufgrund der durch diese Befehle veränderten Position des Graphikcursors wird dieser erneut auf die Position "x/y" gesetzt und zur Eingabeschleife gesprungen (Zeile 360-370).

Besondere Erwähnung verdient die Zeile 280: Wenn die Taste "Esc" (ASCII-Code 27) betätigt wurde, wird kein Unterprogramm aufgerufen, sondern die Graphik wird ausgeschaltet und das Malprogramm beendet.

Cursorsteuerung:

Die Zeilen 390-490 enthalten das Unterprogramm für die Rechtsbewegung des Fadenkreuzes. Das Fadenkreuz wird nur dann nach rechts bewegt, wenn der aktuelle Wert der X-Koordinate kleiner als 317 ist (Zeile 400), da durch die folgende Bewegung um zwei Punkte sonst die rechte Bildschirmbegrenzung überschritten wird (maximale X-koordinate: 319).

Wenn entweder der Lösch- oder aber der Malmodus eingeschaltet ist, wird eine Linie von der momentanen Position des Graphikcursors zur neuen Position gezogen (Zeile 410). Dieses Ziehen einer Linie bedeutet nicht unbedingt, daß die entsprechenden Punkte gesetzt werden: Wenn der Löschmodus eingeschaltet ist, "mode" den Wert null besitzt, werden alle Punkte gelöscht, die sich auf dieser Linie befinden.

Der Graphikcursor wird nun auf die neue Koordinate gesetzt. Der Y-Wert bleibt unverändert, der X-Wert wird um zwei erhöht. Die Erhöhung der X-Koordinate um zwei bedeutet, daß das Fadenkreuz um zwei Punkte nach rechts verschoben wird. Der Grund besteht in der zu geringen Geschwindigkeit, mit der sich das Fadenkreuz ansonsten über den Bildschirm bewegen würde. Die Erhöhung der X-Koordinate um zwei bewirkt, daß es sich bei Betätigung der Taste "Cursor rechts" in Zwei-Punkt-Schritt über den Bildschirm bewegt.

Entsprechendes gilt für alle übrigen Cursortasten. Sie werden nur ausgeführt, wenn das Fadenkreuz innerhalb der Bildschirmbegrenzung bleibt. Von der alten zur neuen Position wird eine Linie gezogen, wenn der "Lösch-Modus" oder der "Mal-Modus" eingeschaltet ist.

Im "Mal-Modus" werden hierbei Punkte gesetzt, im "Lösch-Modus" Punkte, die sich auf dieser Linie befinden, gelöscht.

Der Graphikcursor wird auf die neue Koordinate gesetzt und den Abschluß bildet die Rückkehr zum Hauptprogramm.

Modus umschalten:

Das Umschalten des Modus (Zeile 630-700) geschieht, indem "mode" um eins erhöht wird. Sollte der sich ergebende Wert größer als zwei sein, erhält "mode" den Wert null. Abhängig vom neuen Wert von "mode" wird "mode$" definiert, ein String, der Informationen über den aktuellen Modus enthält. Dieser String wird in der untersten Bildschirmzeile ausgegeben und anschließend zum Hauptprogramm zurückgekehrt.

Parameterpunkt setzen/löschen:

Wenn der Parameterpunkt gesetzt wird, ist dessen Koordinate gleich der aktuellen Koordinate des Graphikcursors. Diese Koordinate wird in "xp" und "yp" zwischengespeichert (Zeile 730), an der jeweiligen Position ein Punkt gezeichnet (Zeile 740) und das Unterprogramm verlassen (Zeile 750).

Wenn der Parameterpunkt gelöscht werden soll, wird der Punkt an der gemerkten Position "xp/yp" gelöscht und "xp" erhält - wir werden noch sehen, aus welchem Grund - den Wert minus eins (Zeile 780-790). Da durch den "DRAW"-Befehl die aktuelle Position des Graphikcursors verändert wird, wird dieser erneut gesetzt, bevor die Rückkehr zum Hauptprogramm erfolgt (Zeile 800-810).

Linie ziehen/löschen:

Wenn der Parameterpunkt gelöscht wurde, das heißt wenn gilt "xp=-1", wird keine Linie gezogen (Zeile 830). Damit soll das versehentliche Anwählen dieser Funktion vermieden werden.

Wenn ein anderer als der "Bewegungs-Modus" eingeschaltet ist, wird eine Linie von der aktuellen Position des Graphikcursors zum Parameterpunkt gezogen. Je nach Modus werden hierbei Punkte gesetzt oder aber gelöscht (Zeile 840).

Durch den "DRAW"-Befehl wird die Position des Graphikcursors verändert und dieser daher anschließend erneut gesetzt (Zeile 850).

Da das Ziehen einer Linie nur dann nicht möglich ist, wenn der Parameterpunkt gelöscht wurde, ergibt sich folgende Möglichkeit: Sie können einen "Stern" malen, indem Sie einen Parameterpunkt angeben, den Cursor wegbewegen und eine Linie ziehen, den Cursor auf einem gedachten Kreis weiterbewegen, dessen Mittelpunkt der Parameterpunkt ist, und weitere Linien ziehen. Dies ist möglich, da - außer er wird absichtlich mit "F4" gelöscht - der Parameterpunkt, das heißt der Kreismittelpunkt, erhalten bleibt.

Rechteck malen:

Das Malen eines Rechtecks (Zeile 880-910) ist mit der Funktion "Linie ziehen" praktisch identisch, abgesehen davon, daß anstelle des Befehls "DRAW" der Befehl "BOX" verwendet wird.

Fläche füllen/löschen:

Diese Funktion besteht praktisch nur aus einer einzigen Zeile (Zeile 940), in der der "PAINT"-Befehl mit den aktuellen Ko-

ordinaten des Graphikcursors ausgeführt wird, wenn entweder der "Mal-Modus" oder der "Lösch-Modus" eingeschaltet ist.

4.2.2. Laufschrift

Wußten Sie schon, daß ihr PLUS/4 mehrere Programme (fast!) gleichzeitig bearbeiten kann? Wenn nicht, empfehle ich Ihnen, das Programm "Laufschrift" einzugeben. Dieses Programm ermöglicht es, einen beliebigen Text in der obersten Bildschirmzeile von rechts nach links "laufen" zu lassen.

Das Besondere daran ist, daß Sie mit dem PLUS/4 währenddessen beliebige Arbeiten ausführen können. Sie können Daten laden oder abspeichern, Programmieren oder mit Programmen arbeiten. Das Programm "Laufschrift" arbeitet gewissermaßen im "Hintergrund". Ohne die über den Bildschirm wandernde Schrift würden Sie niemals auf die Idee kommen, daß ständig ein Programm abläuft.

Das Geheimnis heißt "Interrupt-Programmierung". Diese Programmiertechnik ist jedoch nur in Maschinensprache zu verwirklichen. Sollten Sie bereits entsprechende Kenntnisse besitzen, empfehle ich Ihnen das genaue Studium des Programmablaufs.

Programmbedienung

Nach dem Starten des Programms wird auf Gross/Kleinschrift geschaltet, da das Programm nur in diesem Modus arbeiten kann, ohne Graphikzeichen statt des gewünschten Textes darzustellen.

Danach werden Sie nach dem Text gefragt, der als Laufschrift erscheinen soll, und nach der Geschwindigkeit.

Der Text kann maximal eine Zeile lang sein. "Geshiftete" Zeichen sollten Sie nicht eingeben, da diese als Graphikzeichen dargestellt werden. Auf die Frage nach der Geschwindigkeit

können Sie einen beliebigen Wert zwischen eins und 255 eingeben. Je höher der Wert, desto langsamer ist die Laufschrift. Ich empfehle Ihnen die Geschwindigkeit zwei, bei der sich die Laufschrift gut lesbar und völlig ruckelfrei über den Bildschirm bewegt.

Nach Beantwortung der beiden Fragen wird die Laufschrift gestartet und das BASIC-Programm beendet. Sie können nun beliebige Arbeiten mit dem PLUS/4 durchführen, nur die Belegung der Funktionstasten "F1"-"F3" sollten Sie nicht verändern, da sich dadurch auch die Laufschrift ändern würde. Das nicht mehr benötigte BASIC-Programm kann gelöscht werden, die Laufschrift bleibt dennoch erhalten.

Die Funktionstaste "F7" wurde durch das BASIC-Programm so belegt, daß Sie die Laufschrift mit dieser Taste beliebig aus- und wieder einschalten können.

Der Schrifttext kann jederzeit durch Umbelegung der Funktionstaste "F2" geändert werden, zum Beispiel mit dem Befehl:

`key 2,"laufschrift-demo"` oder `key 2,"test"`

Auch die Geschwindigkeit kann jederzeit variiert werden, indem der gewünschte Wert in die Speicherstelle 218 "gepokt" wird, zum Beispiel:

`poke 218,20` oder `poke 218,100`

Programmlisting

```
100 rem -----laufschrift-----
110 rem -(jan 1986/s.baloui)-
120 :
130 rem ---initialisierung---
140 print chr$(14):rem gross/klein
```

```
150 sa=1630:rem startadresse
160 read a:if a<>-1 then poke sa,a:sa=sa+1:goto 160
170 for i=1 to 7:key i,"":next
180 key 8,"sys 1630"+chr$(13):rem 'help' als
ein/ausschalter
190 sp$="                                        ":rem 40
spaces
200 key 1,sp$:key 3,sp$
210 :
220 rem ---laufschr.parameter---
230 input "text";t$:key 2,t$
240 input "geschwindigkeit";g:poke 218,g
250 print "schalten mit 'help'"
260 sys 1630:rem einschalten
270 end
280 :
290 rem ---datas---
300 data120,173,21,3,201,206,208,20,169,134,141,20,3,169
310 data6,141,21,3,169,0,133,216,165,218,133,217,208,10
320 data169,14,141,20,3,169,206,141,21,3,88,96,198,217
330 data208,36,165,218,133,217,160,0,169,40,24,109,96,5
340 data197,216,176,4,169,0,133,216,166,216,189,103,5,153
350 data0,12,232,200,192,40,208,244,230,216,76,14,206,-1
```

Programmablauf

Das Verständnis des Programmablaufs setzt leider entsprechende Maschinensprachekenntnisse voraus. Sollten Sie diese besitzen, lesen Sie die folgenden Ausführungen bitte sehr sorgfältig, da Kenntnisse über die Interruptprogrammierung des PLUS/4 vermittelt werden, mit der sich - siehe vorliegendes Programm - geradezu an Magie grenzende Phänomene programmieren lassen, die für den reinen BASIC-Programmierer völlig unverständlich sind.

Der PLUS/4 besitzt einen sogenannten "Systeminterrupt" (im folgenden auch "IRQ" genannt), der von den im Rechner enthaltenen Timern jede sechzigstel Sekunde ausgelöst wird. In der Zeropage befindet sich ein Zeiger oder auch "Vektor"

($0314/$0315), der die Adresse der Interruptroutine enthält (normalerweise $CE0E).

Wenn ein Interrupt ausgelöst wird, wird zu der Adresse gesprungen, auf die dieser Zeiger weist. Ab dieser Adresse befinden sich mehrere Maschinenroutinen, die zum Beispiel die STOP-Taste abfragen (denken Sie daran, daß ein laufendes BASIC-Programm in jedem Moment durch Drücken von STOP unterbrochen werden kann, diese Taste daher ständig abgefragt werden muß!).

Das vorliegende Programm basiert auf dem Gedanken, den Interruptvektor zu "verbiegen", so daß er auf eine eigene Routine weist, die dadurch regelmäßig jede sechigstel Sekunde angesprungen wird.

Um weiterhin die Ausführung der eigentlichen Interruptfunktionen zu gewährleisten, muß selbstverständlich am Ende der eigenen IRQ-Routine ein Sprung zur originalen Routine an Adresse $CE0E erfolgen.

Wenn Sie das Listing des Maschinenprogramms am Ende dieses Kapitels betrachten, werden Sie feststellen, daß die eigentliche Routine ab $0686 beginnt. Der Interruptvektor muß daher auf diese neue Adresse verstellt werden.

Diese Aufgabe übernimmt das Initialisierungsprogramm ab $065E. Zuerst werden Interrupts mit dem Befehl "SEI" verhindert. Der Interrupt muß ausgeschaltet sein, wenn der IRQ-Vektor verbogen wird, sonst könnte folgender Fall auftreten: Eines der beiden Bytes des Zeigers wurde soeben verstellt. Nun wird durch die Timer ein Interrupt ausgelöst. Da das zweite Byte noch den Originalwert enthält, wird zu einer undefinierten Adresse gesprungen und es ist sehr wahrscheinlich, daß sich der Rechner nun "aufhängt".

Anschließend wird überprüft, ob der Interruptvektor bereits auf die eigene Routine zeigt, daß heißt ob er bereits verbogen wurde. Dies ist der Fall, wenn das High-Byte des Vektors ($0315) nicht mehr den Originalwert $CE enthält

(Standardinterruptroutine ab $CE0E). In diesem Fall erhält der IRQ-Vektor wieder die Originalwerte $CE und $0E. Unsere eigene Routine ist damit nicht mehr in den Systeminterrupt eingebunden, sie wurde ausgeschaltet.

Enthält $0315 hingegen den Originalwert $CE, so soll unsere Routine eingeschaltet werden. Der IRQ-Vektor wird auf unsere eigene Routine verbogen ($0314/$0315 erhalten die Startadresse unserer Routine, $06/$86). Weiterhin wird $D8 mit dem Wert null initialisiert, und nach $D9 wird der in $DA enthaltene Wert übertragen. Der Sinn dieser Maßnahmen wird erst später deutlich werden. Nach dem Verbiegen des Zeigers wird zum Ende der Initialisierung gesprungen.

Der Befehl "BNE $0684" wirkt wie der unbedingte Sprungbefehl "JMP $0684", da das Zeroflag zuletzt durch "LDA $DA" gesetzt wurde ($DA enthält einen Wert für die gewünschte Geschwindigkeit zwischen eins und 255). Der relative Sprung hat den Vorteil, ein Byte kürzer zu sein als der absolute Sprungbefehl. Dieser "Trick" gehört zum Standardrepertoire jedes Assemblerprogrammierers, es muß jedoch unbedingt darauf geachtet werden, daß der gewünschte Zustand des betreffenden Flags (in diesem Fall des Zero-Flags) in jedem Fall vorhanden ist. Nur dann darf ein absoluter Sprung durch einen relativen ersetzt und damit ein Byte eingespart werden.

Unsere eigene Routine wurde nun entweder ein- oder ausgeschaltet. Der Aufruf des Initialisierungsprogramms wirkt wie ein Umschalter. Interrupts werden nun wieder zugelassen und es folgt die Rückkehr nach BASIC durch "RTS".

Die eigentliche IRQ-Routine basiert auf mehreren Voraussetzungen. Vor der Einbindung der Routine in den Systeminterrupt muß die Funktionstaste "F2" mit dem Text der Laufschrift belegt worden sein. "F1" und "F3" müssen mit je 40 Leerzeichen belegt worden sein. In $DA befindet sich die gewünschte Geschwindigkeit (eins bis 255). Diese notwendige Initialisierung wird durch das BASIC-Programm vorgenommen.

Zum Verständnis der Routine müssen Sie wissen, daß der Bildschirmspeicher des PLUS/4 ab $0C00 beginnt. $0C00 entspricht der ersten Spalte der ersten Zeile, $0C01 der zweiten Spalte der ersten Zeile und so weiter.

Weiterhin müssen Sie wissen, daß die Speicherstellen $055F-$0566 die Längen der Texte enthalten, mit denen die Funktionstasten belegt sind. $055F enthält die Länge des Textes, mit dem "F1" belegt ist, $0560 die Textlänge von "F2" und $0566 die Textlänge von "F8" ("Help").

Der Text, mit dem die Funktionstasten belegt sind, wird vom PLUS/4 ab Adresse $0567 gespeichert und sieht nach der Initialisierung wie folgt aus:

```
$0567: (40 Leerzeichen)(Text)(40 Leerzeichen)
```

Bei jedem ausgelösten Interrupt soll unsere Routine diesen Text in der obersten Bildschirmzeile ausgeben, und zwar jeweils 40 Zeichen (eine Bildschirmzeile = 40 Zeichen).

Zuerst soll der Text ab $0567 ausgegeben werden, das heißt es werden 40 Leerzeichen auf den Bildschirm geschrieben. Die nächste Ausgabe soll ab $0568 erfolgen, wodurch 39 Leerzeichen und das erste Zeichen des eigentlichen Textes ausgegeben werden. Die dritte Ausgabe erfolgt ab $0569 und betrifft dadurch 38 Leerzeichen und zwei Zeichen des Textes, mit dem "F2" belegt wurde.

Der Text scheint durch diese Form der Ausgabe von links nach rechts über den Bildschirm zu wandern. Unsere Routine benötigt einen Zeiger, der angibt, ab welchem Byte der Text momentan auszugeben ist. Für diesen Zeiger wird die Speicherstelle $DA verwendet, die durch die Initialisierungsroutine den Startwert null erhielt.

Betrachten Sie nun die IRQ-Routine. Zuerst wird der Geschwindigkeitszähler $D9 um eins vermindert. Wenn er da-

nach nicht den Wert null enthält, wird zum Ende der IRQ-Routine gesprungen, die Laufschrift wird nicht ausgegeben.

Der Zähler wird bei jedem Aufruf der Routine (jede sechzigstel Sekunde) dekrementiert. Enthielt der Zähler zum Beispiel den von mir empfohlenen Wert zwei, enthält er beim zweiten Aufruf den Wert null, das heißt die Laufschrift wird bei jedem zweiten ausgelösten IRQ (jede dreißigstel Sekunde) ausgegeben.

Wenn der Wert null erreicht wurde, wird der originale vom Benutzer beziehungsweise dem BASIC-Programm in die Speicherstelle $DA "gepokte" Wert wieder in den Zähler $D9 kopiert.

Das Y-Register wird nun mit dem Wert null geladen und der Akkumulator mit dem Wert $28 (dezimal 40). In Speicherstelle $0560 befindet sich die Länge des Textes, mit dem die Funktionstaste "F2" belegt ist. Dieser Wert wird zum Akkumulatorinhalt addiert. Nun wird verglichen, ob unser Positionszähler $D8, ab dem der Text ausgegeben wird, bereits größer ist als die Länge des Textes plus 40.

Wenn $D8 größer ist, weist der Positionszähler hinter den eigentlichen Text, auf die Leerzeichen, mit denen "F3" belegt wurde. Der Text wanderte bereits über den ganzen Bildschirm und der Positionszähler wird mit dem Wert null neu initialisiert.

Das X-Register wird nun mit dem Wert geladen, den der Positionszähler besitzt, und das Zeichen auf dem Bildschirm ausgegeben, das sich an dieser Position ($0567+X) befindet.

Da Y den Wert null besitzt, wird das Zeichen an der Bildschirmadresse $0C00+0=$0C00 ausgegeben, also der ersten Spalte der obersten Zeile. X und Y werden um je eins erhöht und das zweite Zeichen ($0567+1) wird in der zweiten Spalte ($0C00+1) ausgegeben. Dieser Vorgang wiederholt sich, bis Y den Wert $28 (dezimal 40) besitzt, das heißt bis 40 Zeichen ausgegeben wurden.

Den Abschluß der Routine bildet das Erhöhen des Positionszählers mit "INC $D8" und der Sprung zur Originalroutine ($CE0E).

Zum besseren Verständnis dieser zugegebenermaßen nicht ganz einfachen Routine bitte ich Sie, das Programm "Laufschrift" einzugeben und zu starten. Betrachten Sie das Listing des Maschinenprogramms, während sich die Laufschrift bewegt. Sie können die einzelnen Programmschritte weitaus besser nachvollziehen, wenn Sie gleichzeitig die Auswirkungen des Programms vor sich sehen.

Initialisierung:

```
. 065e  78        sei
. 065f  ad 15 03  lda $0315
. 0662  c9 ce     cmp #$ce
. 0664  d0 14     bne $067a
. 0666  a9 86     lda #$86
. 0668  8d 14 03  sta $0314
. 066b  a9 06     lda #$06
. 066d  8d 15 03  sta $0315
. 0670  a9 00     lda #$00
. 0672  85 d8     sta $d8
. 0674  a5 da     lda $da
. 0676  85 d9     sta $d9
. 0678  d0 0a     bne $0684
. 067a  a9 0e     lda #$0e
. 067c  8d 14 03  sta $0314
. 067f  a9 ce     lda #$ce
. 0681  8d 15 03  sta $0315
. 0684  58        cli
. 0685  60        rts
```

IRQ-Routine:

```
. 0686  c6 d9     dec $d9
. 0688  d0 24     bne $06ae
. 068a  a5 da     lda $da
. 068c  85 d9     sta $d9
. 068e  a0 00     ldy #$00
. 0690  a9 28     lda #$28
. 0692  18        clc
. 0693  6d 60 05  adc $0560
. 0696  c5 d8     cmp $d8
. 0698  b0 04     bcs $069e
. 069a  a9 00     lda #$00
. 069c  85 d8     sta $d8
. 069e  a6 d8     ldx $d8
. 06a0  bd 67 05  lda $0567,x
. 06a3  99 00 0c  sta $0c00,y
. 06a6  e8        inx
. 06a7  c8        iny
. 06a8  c0 28     cpy #$28
. 06aa  d0 f4     bne $06a0
. 06ac  e6 d8     inc $d8
. 06ae  4c 0e ce  jmp $ce0e
```

4.2.3. Funktionsdarstellung

Mit dem vorliegenden Programm können Funktionen mit einer unabhängigen Variablen dargestellt werden.

Beispiele:

1. y=2*x
2. y=sin(x)+cos(x)
3. y=x*2+cos(x*x)

Die betreffende Funktion kann in jedem beliebigen Wertebereich dargestellt werden. Dieses Programm dürfte vorwiegend

für Schüler interessant sein, da sich anhand der graphischen Darstellung mit einem Blick in etwa feststellen läßt, ob errechnete nullstellen, Extrema oder Wendepunkte korrekt ermittelt wurden.

Programmbedienung

Nach dem Starten des Programms erhalten Sie eine Kurzanleitung, wie die gewünschte Funktion einzugeben ist. Auf dem Bildschirm erscheint die Zahl 140 und der Cursor befindet sich auf der ersten Stelle dieser Zahl, der eins.

Bewegen Sie den Cursor mindestens drei Spalten nach rechts und geben Sie die gewünschte Funktion ein. Auf dem Bildschirm sollte sich nun zum Beispiel folgendes Bild ergeben:

```
140 y=sin(x)+cos(x)
```

Drücken Sie "F1", um die Eingabe der Funktion zu beenden. Sie werden nun nach dem darzustellenden Wertebereich gefragt, nach dem Anfangs- und dem Endwert des gewünschten Intervalls. Geben Sie zum Beispiel ein:

```
anfangswert fuer x? -10
endwert fuer x? 10
```

Die letzte Frage lautet "Schrittweite (1-10)?". Geben Sie auf diese Frage einen Wert zwischen eins und zehn ein. Je größer der gewählte Wert ist, umso gröber wird die Funktion gezeichnet werden, je kleiner der Wert ist, desto feiner wird sie dargestellt.

Falls Sie nun der Ansicht sind, daß es keinen Grund gäbe, nicht immer die feinste Funktionsdarstellung zu wählen, muß ich Sie enttäuschen: Graphische Funktionsdarstellungen und die dazu nötigen Berechnungen sind Aufgaben, die Ihren PLUS/4 bis an

die Grenzen seiner Leistungsfähigkeit fordern. Sie werden dies an der nicht gerade berauschenden Geschwindigkeit sehen, mit der vor allem komplizierte Funktionen Punkt für Punkt gezeichnet werden.

Je geringer die gewählte Schrittweite ist, desto langsamer wird die Zeichengeschwindigkeit. Geben Sie daher wann immer möglich als Schrittweite eine zehn ein. Sollte sich diese Auflösung als zu gering erweisen, um die betreffende Funktion exakt darzustellen, können Sie die Schrittweite in einem weiteren Durchgang verringern.

Nach Eingabe der Schrittweite beginnt das Programm mit den nötigen umfangreichen Berechnungen. Ihnen bleibt nun eine kleine Wartepause - abhängig von der Komplexität der eingegebenen Funktion -, bevor die hochauflösende Graphik eingeschaltet und die Funktion gezeichnet wird.

Außer der eigentlichen Funktion wird ein Koordinatenkreuz gemalt und dieses skaliert. Die Endpunkte der X-Achse entsprechen den von Ihnen gewählten Anfangs- und Endwerten des darzustellenden Funktionsabschnittes.

Die Einteilung der Y-Achse wird entsprechend den errechneten Y-Werten im vorgegebenen Intervall so gewählt, daß der kleinste beziehungsweise größte Y-Wert gerade noch auf den Bildschirm paßt. Sie sehen daher immer den vollständigen Funktionsverlauf innerhalb des gewählten Intervalls.

Nachdem die Funktion ausgegeben wurde, wartet das Programm auf einen Tastendruck, um anschließend von neuem zu starten.

Beispiel für eine Funktionseingabe mit allen benötigten Parametern:

```
FUNKTION EINGEBEN, Z.B.:  140 Y=2*X
UND 'F1' DRUECKEN



140 Y=2*X
RUN 330
ANFANGSWERT FUER X?  -10
ENDWERT FUER X?  10
SCHRITTWEITE (1-10)?  10
...
...
```

Achtung! Verändern Sie keinesfalls beim Abtippen des Programms die Zeilennummern. Selbst wenn Sie die Sprünge (GOTO, GOSUB) entsprechend ändern, wird das Programm nicht korrekt arbeiten!

Programmlisting

```
100 rem -----funktionsdarstellung-----
110 rem ------(jan 1986/s.baloui)-----
120 :
130 goto 180
140 y=x^2*cos(x)
150 return
160 :
170 rem ---variablen---
180 tp=1319:rem tastaturpuffer
190 tz=239:rem tastaturpufferzaehler
```

```
200 co$=chr$(145):rem 'cursor oben'
210 :
220 rem ---funktionseingabe---
230 scnclr
240 print "funktion eingeben, z.b.:    140 y=2*x"
250 print "und 'f1' druecken"
260 print:print:print
270 print "140";:rem 'cursor home'
280 print co$co$co$
290 key 1,chr$(13)+"run 330"+chr$(13)
300 end
310 :
320 rem ---ausgabe vorbereiten---
330 input "anfangswert fuer x";xa
340 input "endwert fuer x";xe
350 input "schrittweite (1-10)";sw
360 :
370 sx=(xe-xa)/319
380 :
390 for i=0 to 319
400 x=xa+i*sx
410 gosub 140
420 if y>yg then yg=y
430 if y<yk then yk=y
440 next
450 :
460 if abs(yk)>yg then ym=yk:else ym=yg
470 sy=100/ym
480 :
490 rem ---koordinatenkreuz---
500 graphic 1,1
510 :
520 char 1,0,0,str$(ym)
530 char 1,0,13,str$(xa)
540 char 1,0,24,str$(-ym)
550 char 1,40-len(str$(xe)),13,str$(xe)
560 :
570 draw 1,0,100 to 319,100
580 draw 1,1,0 to 3,199
590 :
```

```
600 rem ---funktion zeichnen---
610 for i=0 to 319
620 for ii=1 to 10 step sw
630 x=(xa+i*sx+(sx/10*ii))
640 gosub 140:rem y berechnen
650 y=100-y*sy
660 draw 1,i,y
670 next ii,i
680 :
690 getkey a$
700 graphic 0
710 run
```

Programmablauf

Die darzustellende Funktion befindet sich in Zeile 140 in Form eines Unterprogramms. Dieses Unterprogramm steht am Programmanfang, da es zur Berechnung jedes einzelnen Punktes aufgerufen werden muß. Bei jedem Aufruf eines Unterprogramms ("GOSUB 140") wird die entsprechende Zeilennummer vom BASIC-Interpreter ab dem Programmanfang gesucht. Je weiter vorn im Programmm sich die gesuchte Zeilennummer befindet, desto kürzer ist die Suchzeit. Stellen Sie extrem häufig verwendete Unterprogramme daher bitte möglichst an den Anfang eines Programms.

Der Bildschirm wird nun gelöscht (Zeile 230), der Benutzer erhält Informationen zur Programmbedienung und die Funktionstaste "F1" wird mit der Zeichenkette 'chr$(13)+"run 330"+chr$(13)' belegt (Zeile 290), bevor das Programm beendet wird.

Am Anfang der momentanen Cursorzeile befindet sich die Zeilennummer 140 (siehe Zeile 270-280). Der Benutzer gibt nun die Funktionsgleichung im Direktmodus ein und drückt zum Abschluß der Eingabe "F1". Durch das erste Zeichen, mit dem diese Taste belegt wurde (chr$(13)=ASCII-Code von "Return"), wird die neue Zeile 140 in das Programm übernommen.

Danach wird durch die weitere Belegung dieser Funktionstaste "RUN 330" ausgegeben und ausgeführt. Das Prinzip der Funktionseingabe beruht demnach auf der Änderung von Zeile 140, die die Gleichung enthält, und dem anschließenden Neustart des Programms mit Zeile 330.

Der Benutzer wird nach den Grenzen "xa" und "xe" des gewünschten Wertebereichs und der Schrittweite "sw" gefragt (Zeile 330-350), danach wird ein Faktor "sx" berechnet, der die Unterteilung des Intervalls gemäß der Unterteilung der X-Achse (maximal 320 Punkte in der hochauflösenden Graphik) angibt.

Im nächsten Schritt wird der größte beziehungsweise kleinste in dem gewünschten Wertebereich auftretende Y-Wert berechnet. Das Intervall wird (gemäß der Unterteilung der X-Achse des Graphikbildschirms) in 320 Punkte unterteilt. Zu jedem der 320 X-Werte wird der zugehörige Y-Wert ermittelt und mit dem bisher kleinsten "yk" beziehungsweise grösstem "yg" Y-Wert verglichen (Zeile 420-430). Nach 320 Schleifendurchgängen enthalten "yg" und "yk" den kleinsten und den größten in dem Intervall auftretenden Y-Wert (unter der Bedingung, daß zwischen den 320 Abtastschritten nicht "zwischendurch" extreme Werte auftreten!).

In Zeile 460 wird der absolut größte dieser beiden Werte "ym" ermittelt und in Zeile 470 daraus der Streckungsfaktor "sy" berechnet. Dieser Streckungsfaktor wird so gewählt, daß alle Y-Werte auf dem Bildschirm gezeichnet werden können, die Kurve somit nicht teilweise außerhalb des Bildschirms verläuft. Bedenken Sie, daß nur 200 Punkte zur Verfügung stehen, die Y-Koordinate null sich in der Bildschirmmitte befinden soll, und daher noch je 100 Schritte in der Y-Richtung zur Darstellung positiver und negativer Y-Werte im gewählten Intervall zur Verfügung stehen.

Die hochauflösende Graphik wird eingeschaltet (Zeile 500) und ein Koordinatenkreuz gezeichnet (Zeile 570-580) und skaliert, das heißt mit den Extremwerten von X und Y versehen (Zeile 520-550).

Nach diesen Vorbereitungen folgt die Ausgabe der Funktion (Zeile 610-670) in einer Schleife mit 320 Durchgängen entsprechend der Graphikauflösung von 320 Punkten in der X-Richtung (Zeile 610). Eine innere Schleife dient der weiteren Unterteilung der Schrittweite entsprechend dem gewählten Wert eins bis zehn (Zeile 620).

Aus dem gegebenen X-Wert wird der zugehörige Y-Wert berechnet (Zeile 630), gemäß dem Streckungsfaktor "sy" transformiert (Zeile 650), und der Punkt gezeichnet, dessen X- und Y-Koordinate nun festgelegt ist (Zeile 660).

Wenn diese Ausgabeschleife beendet ist, das heißt die Funktion in dem gewählten Intervall komplett ausgegeben wurde, wird auf eine Taste gewartet, die Graphik ausgeschaltet, und das Programm mit "RUN" neu gestartet (Zeile 690-710).

4.3. Text- und Dateiverarbeitung

Die folgenden drei Programme verwenden Dateien zur dauerhaften Speicherung von Daten. Die Datenspeicherung wurde für die Zusammenarbeit mit einem Diskettenlaufwerk zugeschnitten. Im Kapitel "Programmänderungen bei Cassettenbetrieb" finden Sie Hinweise, wie eine Umstellung dieser Programme für den Betrieb mit der Datasette problemlos möglich ist. Es ändern sich jeweils nur wenige Programmzeilen, das eigentliche Hauptprogramm bleibt unverändert.

4.3.1. Textverarbeitung/Dateiverwaltung

Das vorliegende Programm stellt zweifellos einen Höhepunkt dieses Buches dar: Es handelt sich um eine kombinierte Textverarbeitung und Dateiverwaltung.

Der große Vorteil einer Textverarbeitung gegenüber der herkömmlichen Schreibmaschine dürfte bekannt sein: Tippfehler können auch ohne "Tipp-Ex" problemlos korrigiert, Texte können abgespeichert und zu jedem späteren Zeitpunkt wieder ge-

laden werden, und vor allem können auch nach dem Ausdruck entdeckte Fehler jederzeit behoben und der Text erneut ausgedruckt werden.

Eine Textverarbeitung ist jedoch zugleich auch - fast! - eine Dateiverwaltung: Stellen Sie sich vor, jede Textzeile enthalte einen Datensatz, zum Beispiel eine komplette Adresse. Die wesentlichen Anforderungen, die an eine Dateiverwaltung gestellt werden, werden dann ebenfalls von der Textverarbeitung erfüllt:

Eintragen eines neuen Datensatzes entspricht dem Beschreiben einer neuen Textzeile mit dem jeweiligen Datensatz. Löschen eines Datensatzes entspricht dem Löschen der betreffenden Textzeile, und Ändern eines Satzes wird durch Änderungen des Textes der jeweiligen Zeile sogar problemloser gelöst als in mancher reinen Dateiverwaltung.

Sogar das Suchen nach bestimmten Datensätzen kann problemlos gelöst werden mit Hilfe der Funktion "Suche nach bestimmten Textteilen", die in fast allen Textverarbeitungsprogrammen vorhanden ist. Diese Funktion wurde im vorgestellten Programm an die spezifischen Bedürfnisse einer Dateiverwaltung angepaßt.

Eine Funktion bleibt übrig, die keine Textverarbeitung bietet, ohne die jedoch eine gute Dateiverwaltung schwer vorstellbar ist: Das Sortieren von Datensätzen. Diese textverarbeitungsunspezifische Funktion wurde in diesem Programm eingebaut, das damit gleichermaßen als Textverarbeitung und als Dateiverwaltung zu verwenden ist.

Programmbedienung

Da dieses Programm eine Vielfalt verschiedener Funktionen bietet, ist es leider auch schwieriger als manch anderes Programm zu bedienen, das in diesem Buch vorgestellt wird. Seien Sie bitte beim Abtippen des Listings besonders sorgfältig, da sich aufgrund der Programmlänge eine Vielzahl von Fehlern

einschleichen und jeder einzelne bereits eine kleine "Katastrophe" beim Programmlauf verursachen kann.

Nach dem Starten des Programms sehen Sie einen fast leeren Bildschirm vor sich. Nur die obersten Bildschirmzeilen sind mit Informationen zur Programmbedienung beschrieben. In der ersten Zeile unter diesen Benutzerinformationen sehen Sie einen invers dargestellten Cursor. Dieser zeigt Ihnen ihre momentane Schreibposition an. Sie können sofort anfangen, Text einzugeben.

Innerhalb einer Zeile können Sie mit den Tasten "Del" und "Inst" ein Zeichen löschen beziehungsweise in den Text einfügen. Mit den Cursortasten für rechts/links können Sie sich in einer Zeile bewegen, ohne dabei Zeichen zu verändern.

Wenn Ihnen die Cursorbewegung mit den Tasten "Cursor rechts" und "Cursor links" zu langsam ist, können Sie mit den Tasten "Return" und "Shift Return" (= Return zusammen mit Shift gedrückt) Sprünge um jeweils acht Zeichen nach rechts beziehungsweise links ausführen.

Beim Schreiben werden Sie feststellen, daß nach der Eingabe des 74. Zeichens keine weitere Eingabe mehr möglich ist. Sie befinden sich nun nun am Ende der "Druckzeile". Mit Druckzeile ist gemeint, daß in diesem Programm eine später auszudruckende Zeile die Länge 74 Zeichen besitzt, eine übliche Einstellung für die Schreibbreite bei Verwendung normalen DIN A4-Papiers.

Um in die nächste Zeile zu gelangen, drücken Sie die Taste "Cursor nach unten", das heißt die nach unten gerichtete Pfeiltaste. In der obersten Bildschirmzeile wird die Druckzeile angezeigt, in der Sie sich jeweils befinden (nach dem Starten des Programms in Zeile eins). Ihnen stehen maximal 500 Druckzeilen für Ihren Text zur Verfügung. In einem Durchgang können Sie daher circa zehn DIN A4-Seiten beschreiben.

Es ist jedoch möglich, den bearbeiteten Text abzuspeichern, im Speicher zu löschen, und anschließend einen neuen Text zu schreiben.

Wenn Sie am unteren Bildschirmrand angelangt sind, können Sie mit der obersten Funktionstaste "F1" zur nächsten Bildschirmseite blättern. Sie befinden sich dann in der Druckzeile Nummer 13. Wenn Sie die gleiche Funktionstaste zusammen mit "Shift" betätigen (= "F4"), wird eine Seite zurückgeblättert. Mit diesen Funktionstasten können Sie jederzeit Ihren Text Bildschirmseite für Bildschirmseite durchblättern.

Wenn Sie in einer bereits verlassenen Zeile Fehler entdecken, können sie mit der Taste "Cursor nach oben", das heißt der nach oben gerichteten Cursortaste, problemlos die vorangehende Zeile erreichen und den Fehler korrigieren.

Weitere Editiermöglichkeiten bieten die Funktionstasten "F2" und "F5": Mit "F2" können Sie an der momentanen Position eine Zeile einfügen, mit "F5" können Sie die Zeile löschen, in der Sie sich befinden. Zuvor müssen Sie jedoch die Sicherheitsabfrage "Sind Sie sicher (j/n) ?" durch Drücken von "j" beantworten. Diese Sicherheitsabfrage soll ein versehentliches Löschen von Zeilen verhindern.

Editiermöglichkeiten innerhalb einer Zeile:

1. "Cursor rechts" = Cursor ein Zeichen nach rechts
2. "Cursor links" = Cursor ein Zeichen nach links
3. "Return" = Sprung um acht Zeichen nach rechts
4. "Shift Return" = Sprung acht Zeichen nach links
5. "Del" = Zeichen löschen
6. "Inst" = Zeichen einfügen

Editiermöglichkeiten im Text:

1. "Cursor nach unten" = Cursor zur nächsten Zeile
2. "Cursor nach oben" = Cursor zur vorigen Zeile
3. "F1" = Bildschirmseite weiterblättern
4. "F4" = Bildschirmseite zurückblättern

5. "F2" = Zeile einfügen
6. "F5" = Zeile löschen

Vier weitere Funktionstasten stehen noch zur Verfügung: "F3", "F6", "F7" und "HELP". Diese Taste "HELP" wird zum Abspeichern des Textes benötigt. Wenn Sie sie betätigen, werden Sie gefragt, ob Sie auch wirklich sicher sind. Wenn ja, drücken Sie "j", ansonsten - wenn Sie die Funktion versehentlich angewählt haben - die Taste "n".

Gehen wir davon aus, daß Sie den Text tatsächlich abspeichern wollen und dies mit "j" bestätigen. Der Bildschirm wird nun gelöscht und Sie werden nach dem Namen des Textes gefragt. Unter dem von Ihnen einzugebenden Namen wird der Text auf der Diskette gespeichert (zum Beispiel "DEMO-TEXT"). Diesen Namen müssen Sie angeben, wenn Sie den Text zu einem späteren Zeitpunkt wieder laden wollen.

Beim Abspeichern dürfen Sie sich nun in Geduld üben, je nach Länge des eingegebenen Textes.

Die Taste "F7" hat die entgegengesetzte Funktion, das Laden eines Textes. Überlegen Sie sich die Antwort auf die folgende Sicherheitsabfrage ("j" oder "n") gut, denn beim Laden wird der Text, der sich momentan im Rechnerspeicher befindet, restlos gelöscht. Geben Sie den Namen an, unter dem Sie den zu ladenden Text abspeicherten und warten Sie, bis er auf dem Bildschirm erscheint.

Eine Text- oder Dateiverarbeitung ist nicht vollständig, wenn die gespeicherten Daten nicht ausgedruckt werden können. Der Ausdruck wird mit der Taste "F3" gestartet. Bevor Sie die folgende Sicherheitsabfrage mit "j" beantworten, vergewissern Sie sich, daß der Drucker eingeschaltet und das Papier korrekt justiert ist.

Mit der Taste "F6" können die eingegebenen Daten sortiert werden, eine Funktion, die bei der reinen Textverarbeitung

überflüssig, für die Dateiverwaltung jedoch unbedingt notwendig ist.

Nehmen wir an, Sie verwenden das Programm zur Verwaltung von Adressen. In jede Druckzeile tragen Sie eine Adresse ein. Folgende Adressen wurden bisher von Ihnen eingegeben:

```
Maier/Gerhard/Ottostr. 100 a/6700 Ludwigshafen/0621-12345
Wilbert/Oskar/Friedelweg 5/2000 Ottobrunn/2645-57683
Aarndt/Willi/Bleichweg 2/8000 Stuttgart/7455-25602
Schmidt/Anette/Schmidtweg 76/1000 Berlin/57240-78927
```

Wenn Sie mit der Taste "F6" nun die Funktion "Datensätze sortieren" anwählen, werden die einzelnen Sätze entsprechend ihrer alphabetischen Reihenfolge umgestellt und Sie erhalten folgende sortierte Reihenfolge:

```
Aarndt/Willi/Bleichweg 2/8000 Stuttgart/7455-25602
Maier/Gerhard/Ottostr. 100 a/6700 Ludwigshafen/0621-12345
Schmidt/Anette/Schmidtweg 76/1000 Berlin/57240-78927
Wilbert/Oskar/Friedelweg 5/2000 Ottobrunn/2645-57683
```

Dieser Sortiervorgang ist jederzeit möglich, wenn zumindest zwei Datensätze eingetragen wurden. Die Zeitdauer, die das Sortieren in Anspruch nimmt, hängt von der vorhandenen Datensatzanzahl ab.

Sollte die alphabetische Reihenfolge Ihrer Adressdatei durch Änderungen oder das Eintragen weiterer Datensätze durcheinandergeraten sein, ist mit dieser Funktion eine Neusortierung möglich.

Eine sehr interessante Funktion stellt das Suchen von bestimmten Datensätzen oder Textteilen dar, die mit Hilfe der

Taste "ESC" eingeleitet wird. Wenn Sie diese Taste betätigen, wird der Bildschirm gelöscht und Sie werden aufgefordert, Suchbegriffe einzugeben. Sie können nun eine oder auch mehrere Zeichenketten eingeben, nach denen gesucht werden soll.

Gehen wir zum Beispiel von unserer Beispiel-Adressdatei aus: Sie suchen eine Person, die in Berlin wohnt. Geben Sie als Suchbegriff ein:

Berlin

Der erste Datensatz, in dem der Suchbegriff "Berlin" vorkommt, wird auf dem Bildschirm ausgegeben und Sie werden gefragt, ob weitergesucht werden soll (möglicherweise suchen Sie nicht die Frau "Schmidt" in Berlin, sondern eine andere in Berlin wohnende Person).

Wenn Sie weitersuchen wollen, wird der nächste Datensatz ausgegeben, in dem der Suchbegriff vorkommt (sofern in der Datei vorhanden!).

Wenn Sie auf die Frage "Weitersuchen (j/n) ?" mit "n" antworten, erscheinen die eingegebenen Daten wieder wie gewohnt und der Cursor befindet sich exakt in jener Zeile, in der auch der gefundene Datensatz eingetragen ist.

Sie können nicht nur einen, sondern auch mehrere Suchbegriffe angeben. Mehrere Suchbegriffe müssen durch das Sonderzeichen "^" getrennt werden. Sie suchen zum Beispiel nach:

Maier^Berlin

Ausgegeben werden nur jene Datensätze, in denen beide Suchbegriffe vorkommen, zum Beispiel:

Maier/Willi/Aalweg 5/1000 Berlin/87642-16345

jedoch nicht der Datensatz:

Maier/Anton/Idaweg 5/2000 Hamburg/836787-71752

Übrigens: Sowohl nach dem Starten des Programms als auch bei der Ausführung verschiedener Funktionen wie zum Beispiel "Zeile löschen" oder "Zeile einfügen" müssen Sie mehrere Sekunden Wartezeit in Kauf nehmen. Der Grund hierfür liegt in der Programmiersprache BASIC, die zu langsam ist, um 500 Textzeilen ohne Verzögerungen zu verwalten. Dieses Problem kann nur gelöst werden, indem zeitkritische Programme wie Textverarbeitungen vollständig in Maschinensprache geschrieben werden.

Programmlisting

```
100 rem -----textverarbeitung-----
110 rem --(januar 1986/s.baloui)--
120 :
130 rem ---variablen---
140 rem cs=cursorspalte
150 rem cz=cursorzeile
160 rem cr=cursor-setz-routine
170 rem maxl=max.zeilenanzahl
180 rem lm=max.zeilenlaenge
190 rem lpage=max.seitenzahl
200 rem tab=tabulatorsprungweite
210 rem ft$=zeichen, mit denen die funktionstasten neu
belegt wurden
220 rem cr$-ra$=ascii-codes der steuertasten in stringform
230 rem ed$=tasten, mit denen eine eingabe editiert werden
kann
```

```
240 rem ez$=tasten, mit denen die eingabe in ein feld
beendet wird
250 rem z$=zeichen, die als gueltige eingabe zugelassen
werden
260 rem i(...)=positionen von '^'
270 rem u$(...)=feldinhalte
280 rem po=position im eingabefeld
290 rem lm=laenge des aktuellen feldes
300 rem li=nummmer des aktuellen feldes
310 :
320 rem ---zeilen/seitenzahl---
330 maxl=500
340 dim u$(maxl)
350 lm=74
360 lpage=11
370 pmax=int(maxl/lpage)
380 tab=8
390 :
400 rem ---cursor---
410 cs=202
420 cz=205
430 cr=55464
440 :
450 rem ---funktionstastenbelegung---
460 for i=1 to 8
470 f$(i)=chr$(i+132)
480 ft$=ft$+f$(i)
490 key i,f$(i)
500 next
510 :
520 rem ---steuerzeichen---
530 cr$=chr$(29):rem cursor rechts
540 cl$=chr$(157):rem cursor links
550 cu$=chr$(17):rem cursor unten
560 co$=chr$(145):rem cursor oben
570 ho$=chr$(19):rem cursor home
580 del$=chr$(20):rem taste 'del'
590 inst$=chr$(148):rem taste 'inst'
600 rt$=chr$(13):rem taste 'return'
610 sr$=chr$(141):rem tasten 'shift'+'return'
```

```
620 esc$=chr$(27):rem taste 'esc'
630 re$=chr$(18):rem revers ein
640 ra$=chr$(146):rem revers aus
650 :
660 rem ---versch.strings---
670 ed$=cr$+cl$+del$+inst$
680 ez$=cu$+co$+ft$+rt$+sr$+esc$
690
z$="1234567890!#$%&'()qwertyuiopasdfghjklzxcvbnmQWERTYUIOPA
SDFGHJKLZXCVBNM"
700 z$=z$+" +-*/.,;<>[]?=^"
710 for i=1 to lm:sp$=sp$+" ":next
720 goto 1070:rem hauptprogramm
730 :
740 rem -----eingabe-routine-----
750 print re$ mid$(u$(li),po,1) ra$ cl$;
760 getkey a$
770 if instr(ez$,a$) then print mid$(u$(li),po,1) cl$;:
return
780 on instr(ed$,a$) goto 850,890,930,990
790 if instr(z$,a$)=0 or po>=lm then 750
800 :
810 rem ---normales zeichen---
820 mid$(u$(li),po,1)=a$:po=po+1:print a$;:goto 750
830 :
840 rem ---cursor links---
850 if po<lm then print mid$(u$(li),po,1);:po=po+1
860 goto 750
870 :
880 rem ---cursor rechts---
890 if po>1 then print mid$(u$(li),po,1);:po=po-1:print cl$
cl$;
900 goto 750
910 :
920 rem ---delete---
930 s=peek(cs):z=peek(cz)
940 h$=mid$(u$(li),po+1)+" ":print h$;:mid$(u$(li),po)=h$
950 poke cs,s:poke cz,z:sys cr
960 goto 750
970 :
```

```
980 rem ---insert---
990 if po=lm then 1030
1000 s=peek(cs):z=peek(cz)
1010 h$=" "+mid$(u$(li),po,lm-po):print h$;:u$(li)=
mid$(u$(li),1,po-1)+h$
1020 poke cs,s:poke cz,z:sys cr
1030 goto 750
1040 :
1050 :
1060 rem ---------hauptprogramm-------
1070 print chr$(14):rem gross/klein
1080 for i=0 to maxl:u$(i)=sp$:next
1090 pa=1:zeile=1
1100 scnclr
1110 print ho$;
1120 print re$"Zeile:        (Sh)Return=Tab   Esc=Suche";
1130 print re$"F1/F4=Blaettern     F2/F5=Del/Inst Zeile";
1140 print re$"F3/F6=Druck/Sort. Help/F7=Save/Load     "
1150 :
1160 for i=1 to lpage
1170 poke cs,0:poke cz,2*(i-1)+3:sys cr
1180 print u$((pa-1)*lpage+i);
1190 next
1200 :
1210 li=zeile+(pa-1)*lpage:l$=left$(str$(li)+"    ",4):print
ho$ re$ tab(8)l$
1220 poke cs,0:poke cz,2*(zeile-1)+3:sys cr
1230 po=1
1240 gosub 750
1250 k=instr(ez$,a$)
1260 on k goto 1460,1500,1540,1620,1960,1580,1700,1780,
2060,2220,1300,1380,2380
1270 goto 1240
1280 :
1290 rem ---tabsprung vorwaerts---
1300 if po+tab>=lm then 1210
1310 po=po+tab
1320 if po>40 and po<41+tab then poke cz,peek(cz)+1:poke
cs,po-41:goto 1340
1330 poke cs,peek(cs)+tab
```

```
1340 sys cr
1350 goto 1240
1360 :
1370 rem ---tabsprung rueckwaerts---
1380 if po-tab<1 then 1210
1390 po=po-tab
1400 if po<41 and po>40-tab then poke cz,peek(cz)-1:poke
cs,po:goto 1420
1410 poke cs,peek(cs)-tab
1420 sys cr
1430 goto 1240
1440 :
1450 rem ---naechste zeile---
1460 if zeile<lpage then zeile=zeile+1
1470 goto 1210
1480 :
1490 rem ---vorige zeile---
1500 if zeile>1 then zeile=zeile-1
1510 goto 1210
1520 :
1530 rem ---naechste seite---
1540 if pa<pmax then pa=pa+1:zeile=1
1550 goto 1160
1560 :
1570 rem ---vorige seite---
1580 if pa>1 then pa=pa-1:zeile=1
1590 goto 1160
1600 :
1610 rem ---zeile loeschen---
1620 if pa=pmax and zeile=maxl then 1210
1630 for i=zeile+(pa-1)*lpage to maxl-1
1640 u$(i)=u$(i+1)
1650 next
1660 u$(maxl)=left$(sp$,lm)
1670 goto 1160
1680 :
1690 rem ---zeile einfuegen---
1700 if pa=pmax and zeile=maxl then 1210
1710 for i=maxl to zeile+(pa-1)*lpage+1 step-1
1720 u$(i)=u$(i-1)
```

```
1730 next
1740 u$(zeile+(pa-1)*lpage)=left$(sp$,lm)
1750 goto 1160
1760 :
1770 rem ---zeilen sortieren---
1780 printho$ re$"          Sind Sie sicher (j/n) ?          "
1790 getkey a$
1800 if a$<>"j" then 1930
1810 :
1820 a=maxl
1830 if u$(a)=sp$ then a=a-1:goto 1830
1840 if a<2 then 1930
1850 :
1860 for i=a to 2 step-1
1870 x$=""
1880 for ii=1 to i
1890 if u$(ii)>x$ then x$=u$(ii):z=ii
1900 next ii
1910 s$=u$(i):u$(i)=u$(z):u$(z)=s$
1920 next i
1930 goto 1090
1940 :
1950 rem ---ausdrucken---
1960 for i=maxl to 1 step-1
1970 if u$(i)=sp$ then next
1980 open 4,4,7
1990 for ii=0 to i
2000 print#4,"   "u$(ii)
2010 next
2020 close4
2030 goto 1210
2040 :
2050 rem ---text laden---
2060 printho$ re$"          Sind Sie sicher (j/n) ?          "
2070 getkey a$
2080 if a$<>"j" then 2190
2090 scnclr
2100 for i=1 to maxl:u$(i)=sp$:next
2110 input"Name des Textes";n$
2120 open 1,8,2,n$+",s,r"
```

```
2130 input#1,a
2140 for i=1 to a
2150 for ii=1 to lm
2160 get#1,a$:mid$(u$(i),ii)=a$
2170 next ii,i
2180 close1
2190 goto 1090
2200 :
2210 rem ---text abspeichern---
2220 printho$ re$"          Sind Sie sicher (j/n) ?          "
2230 getkey a$
2240 if a$<>"j" then 2350
2250 scnclr
2260 for a=maxl to 1 step-1
2270 if u$(a)=sp$ then next
2280 input"Name des Textes";n$
2290 open 15,8,15,"s:"+n$:close 15:open 1,8,2,n$+",s,w"
2300 print#1,a
2310 for i=1 to a
2320 print#1,u$(i);
2330 next
2340 close1
2350 goto 1090
2360 :
2370 rem ---suche---
2380 scnclr
2390 print"  Suche nach Textteilen/Datensaetzen"
2400 print"  ----------------------------------"
2410 print:print:print"Geben Sie die Suchbegriffe durch
das"
2420 print"Zeichen '^' getrennt ein."
2430 print"Bsp.: Meier^Gerhard^Muenchen"
2440 print"Beenden der Eingabe mit 'Return'"
2450 print:print
2460 u$(0)=left$(sp$,40):li=0:po=1:lm=39:gosub 750:lm=74
2470 if a$<>rt$ or left$(u$(0),1)=" " then print:print
co$;:goto 2460
2480 :
2490 if right$(u$(0),1)=" " then u$(0)=left$(u$(0),
len(u$(0))-1): goto 2490
```

```
2500 z=0:i(0)=1:u$(0)="^"+u$(0)+"^"
2510 :
2520 i(z+1)=instr(u$(0),"^",i(z)+1)
2530 if i(z+1)<>0 then z=z+1:goto 2520
2540 :
2550 for i=1 to maxl
2560 if u$(i)=sp$ then 2640
2570 for ii=0 to z-1
2580 if instr(u$(i),mid$(u$(0),i(ii)+1,i(ii+1)-i(ii)-1))=0
then 2640
2590 next ii
2600 print:print"Gefunden: "u$(i)
2610 print"Weitersuchen (j/n) ?"
2620 getkey a$
2630 if a$="n" then 2670
2640 next i
2650 goto 2690
2660 :
2670 page=int(i/lpages)+1
2680 zeile=i-(page-1)*11
2690 goto 1100
```

Programmablauf

Den Programmablauf werde ich im folgenden stark gekürzt darstellen, da zur eingehenden Beschreibung dieses umfangreichen Programms ein eigenes Buch notwendig wäre. Ich empfehle Ihnen, die Variablenliste am Programmanfang eingehend zu studieren, da sie sehr zum Verständniß des Programms beiträgt.

In den Zeilen 320-380 werden wichtige Parameter definiert. Die maximale Zeilenlänge wird auf 74 festgelegt, die Zeilenanzahl auf 500 und die Anzahl der Druckzeilen pro Bildschirmseite auf elf. Aus der maximalen Zeilenzahl und der Anzahl Zeilen pro Seite ergibt sich in Zeile 370 der Wert von "pmax", der maximalen Seitenanzahl (Bildschirmseiten).

Die Zeilen 450-500 werden zur Neubelegung der Funktionstasten verwendet. Die einzelnen Funktionstasten werden mit den

Zeichen belegt, die den ASCII-Codes 133 bis 140 entsprechen. Aus diesen Zeichen wird der String "ft$" gebildet (Zeile 480).

Im folgenden Programmblock (Zeilen 520-640) werden - wie bereits bekannt - Strings gebildet, die die benötigten Steuerzeichen für die Cursorsteuerung und den Revers-Modus enthalten.

Das eigentliche Hauptprogramm beginnt ab Zeile 1070. Zuerst werden allen Strings, die die späteren Eingaben aufnehmen sollen ("u$(...)"), Spaces entsprechend der maximalen Zeilenlänge zugewiesen (Zeile 1080).

In Zeile 1090 werden die momentane Cursorposition (Seite eins ("pa=1") und die Zeile eins ("zeile=1")) initialisiert, das heißt diese Parameter erhalten einen definierten Ausgangswert.

Nachdem die Benutzerinformationen auf dem Bildschirm ausgegeben wurden, wird der Cursor auf den Beginn der elf Druckzeilen der jeweiligen Seite gesetzt, und es werden die Inhalte der ersten elf Druckzeilen ausgegeben, die zu Beginn der Programmbenutzung aus jeweils 74 Spaces bestehen (Zeilen 1160-1190).

Diese Ausgabe der momentanen Bildschirmseite wird unter anderem auch von den Funktionen "Seite vor" und "Seite zurück" verwendet, in denen nur die entsprechenden Parameter "zeile" und "page" geändert werden. Die Aufgabe, den Inhalt der jeweils nächsten beziehungsweise vorhergehenden Bildschirmseite auszugeben, übernimmt das hier beschriebene Hauptprogramm.

Die Variable "li" erhält nun - in Abhängigkeit von der Druckzeile auf der jeweiligen Bildschirmseite, der momentanen Bildschirmseite "pa" und der Anzahl der Druckzeilen pro Seite "lpage" - einen Wert zugewiesen, der die Zeile (von eins bis 66) kennzeichnet, in der sich der Cursor befindet. Zu Beginn erhält "li" durch diese Berechnung den Wert eins.

Der Cursor wird auf den Anfang der Druckzeile gesetzt (Zeile 1220), "po" erhält den Wert eins (Zeile 1230), entsprechend der

Cursorposition innerhalb der Druckzeile, und der eigentliche Programmkern, die Eingaberoutine, wird aufgerufen (Zeile 1240).

Die Eingaberoutine umfaßt den Programmblock ab Zeile 740 bis Zeile 1030. In Zeile 750 wird das Zeichen der aktuellen Druckzeile "u$(li)" ausgegeben, "auf" dem sich der Cursor momentan befindet (nach Aufruf der Eingaberoutine auf dem ersten Zeichen). Dieses Zeichen wird durch seine Position "po" innerhalb der Druckzeile gekennzeichnet.

Nachdem eine Taste betätigt wurde, wird untersucht, ob das zugehörige Zeichen im String "ez$" enthalten ist, ob es zum Beispiel eine der acht Funktionstasten war, die betätigt wurde. Wenn ja, wird die Eingaberoutine verlassen und zum Hauptprogramm zurückgekehrt, nachdem die Invers-Darstellung des Zeichens unter dem Cursor wieder rückgängig gemacht wurde (Zeile 770).

Ist das Zeichen hingegen in "ed$" enthalten, handelt es sich um eine Taste zur Editierung der Eingabe innerhalb einer Druckzeile und der entsprechende Programmblock wird angesprungen (Zeile 780).

Wenn das Zeichen weder in "ez$" noch in "ed$" vorhanden war, und es außerdem entweder ebenfalls nicht in "z$" (= zugelassene Eingabezeichen) enthalten oder aber bereits die maximale Eingabelänge erreicht ist, erfolgt keinerlei Aktion und es wird zur Eingabeschleife zurückgekehrt (Zeile 790).

Wurde ein zugelassenes Zeichen eingegeben, und ist die maximale Eingabelänge "lm" noch nicht erreicht, wird das Zeichen "a$" ausgegeben, der Positionszähler "po" erhöht, und das Zeichen in den String der aktuellen Druckzeile ("u$(li)") übertragen (Zeilen 810-820).

Zur Bewegung des Cursors um eine Stelle nach rechts beziehungsweise links genügt die Änderungs des Parameters "po", der die Cursorposition in der aktuellen Druckzeile angibt. Die Funktion "Cursor rechts" (Zeile 840-860) prüft zuerst, ob sich

der Cursor bereits am Ende der Druckzeile befindet. Ist dies nicht der Fall, wird das invers dargestellte Zeichen an der momentanen Cursorposition "normalisiert" und "po", der Positionszähler, um eins erhöht, bevor die Rückkehr zur Eingabeschleife erfolgt, die für die inverse Darstellung des Zeichens an der neuen Cursorposition sorgt.

"Cursor links" (Zeile 880-900) verläuft analog. Wenn sich der Cursor noch nicht am Anfang der Druckzeile befindet, wird das Zeichen unter dem Cursor normalisiert, "po" erniedrigt und zur Eingabeschleife zurückgekehrt.

Zur Durchführung der Funktionen "Zeichen löschen" und "Zeichen einfügen" genügt die Veränderung von Parametern leider nicht, da aufgrund gravierender Veränderungen die Druckzeile auf dem Bildschirm neu ausgegeben werden muß, und die jeweilige Änderung auch im String selbst durchzuführen ist.

Zum Löschen des Zeichens unter dem Cursor (Zeile 920-960) wird zuerst die momentane Cursorposition in "s" und "z" zwischengespeichert. Nun wird der String "h$" gebildet, der sich aus den Zeichen rechts vom Cursor zusammensetzt. Dieser String, in dem das Zeichen unter dem Cursor fehlt, wird ab der momentanen Cursorposition ausgegeben (auf dem Bildschirm wird dadurch das Zeichen unter dem Cursor gelöscht!), und danach der Inhalt der aktuellen Druckzeile korrigiert, der sich in "u$(li)" befindet (Zeile 940).

"u$(li)" enthält anschließend die Zeichen links vom Cursor, die Zeichen rechts vom Cursor und zum Abschluß ein Leerzeichen oder "Space", das heißt den gleichen Inhalt wie zuvor außer jenem Zeichen, das sich zuvor unter dem Cursor befand, und einem Leerzeichen als letztem Zeichen. Zum Schluß der Routine wird der Cursor - dessen Position durch die Bildschirmausgabe geändert wurde - wieder auf die zwischengespeicherte Position gesetzt (Zeile 950).

Das Einfügen eines Leerzeichens verläuft analog (Zeile 980-1030), abgesehen davon, daß diese Funktion nur dann ausge-

führt wird, wenn sich der Cursor nicht am Ende der Druckzeile befindet (Zeile 980).

Die momentane Cursorposition wird gerettet, "h$" aus einem Leerzeichen, dem Zeichen unter und den Zeichen rechts vom Cursor gebildet, dieser String ausgegeben, wodurch alle Zeichen der Druckzeile um eine Spalte nach rechts verschoben werden und an der Cursorposition ein Leerzeichen ausgegeben wird, der Inhalt der Druckzeile "u$(li)" entsprechend manipuliert (ein Leerzeichen an der Position "po" eingefügt), und zuletzt wird der Cursor wieder auf die ursprüngliche Position gesetzt.

Alle "größeren" Aktionen außer dem Editieren eines einzelnen Zeichens und der Bewegung des Cursors werden außerhalb der Eingaberoutine durchgeführt. Wenn die betreffende Kommandotaste in "ez$" enthalten ist, wird die Eingaberoutine verlassen - nachdem das inverse Zeichen unter dem Cursor zuvor normalisiert wurde - und je nach Kommando der entsprechende Programmteil angesprungen (Zeile 1250-1260).

Eine solche "größere" Aktion ist der Tabulatorsprung um acht Spalten vorwärts, das heißt nach rechts (Zeile 1290-1350). Dieser Sprung wird nur dann ausgeführt, wenn die sich ergebende Cursorposition nicht größer als die maximale Druckzeilenlänge "lm" ist (Zeile 1300).

Die Position "po" wird um acht erhöht und anschließend geprüft, ob der Sprung dazu führt, daß sich der Cursor in die nächste Bildschirmzeile bewegt. Wenn ja, wird die neue Cursorspalte ermittelt und die Cursorzeile um eins erhöht (Zeile 1320). Bleibt der Cursor in der gleichen Zeile, wird die momentane Cursorspalte einfach um acht, den Betrag des Sprunges, erhöht.

Der Cursor wird auf die ermittelte Position gesetzt und zu Zeile 1240 gesprungen, in der die Eingaberoutine erneut aufgerufen wird.

Der Tabulatorsprung rückwärts, das heißt um acht Spalten nach links, bietet nichts Neues (Zeile 1370-1430). Er wird nur dann ausgeführt, wenn die sich ergebende Cursorposition nicht kleiner

als eins ist. Ist dies nicht der Fall, wird "po" um acht vermindert, die neue Spalten- und Zeilenposition des Cursors ermittelt, der Cursor auf die ermittelte Position gesetzt, und der Programmteil verlassen.

Interessanter ist die Funktion "nächste Zeile" (Zeile 1450-1470), zu deren Durchführung die Veränderung des Parameters "zeile" ausreicht. Befindet sich der Cursor bereits in der elften Druckzeile der momentanen Bildschirmseite ("zeile=lpage"), wird diese Funktion nicht ausgeführt (Zeile 1460). Ansonsten wird die Variable "zeile" erniedrigt und zu Zeile 1210 gesprungen. Der dort folgende Programmblock errechnet die echte Druckzeile "li", setzt den Cursor und ruft die Eingaberoutine auf.

"Vorige Zeile" (Zeile 1490-1510) wird ausgeführt, wenn sich der Cursor nicht in der ersten Druckzeile der jeweiligen Bildschirmseite befindet (Zeile 1500). Dabei wird "zeile" um eins vermindert.

Um zur nächsten Seite weiterzublättern, wird der Programmblock ab Zeile 1530 verwendet. Dieses Blättern ist nur möglich, wenn die letzte Seite "pmax" noch nicht erreicht wurde (Zeile 1540). Es genügt wiederum die Veränderung mehrerer Parameter: "pa", der Seitenzähler, wird erhöht, und "zeile", der Zähler für die aktuelle Druckzeile auf der jeweiligen Bildschirmseite, wird auf eins gesetzt. Diese Veränderungen bedeuten, daß sich der Cursor nun auf der ersten Druckzeile der folgenden Bildschirmseite befinden soll.

Diese folgende Bildschirmseite muß selbstverständlich komplett ausgegeben werden, alle elf Druckzeilen dieser Seite müssen auf den Bildschirm "geprintet" und der Cursor auf den Anfang der ersten Zeile gesetzt werden. Diese Aufgabe erledigt wie erwähnt das ab Zeile 1160 angesprungene Hauptprogramm, das nach diesen Ausgaben die Eingaberoutine aufruft.

Das Zurückblättern zur vorigen Seite verläuft analog (Zeile 1580-1590): Wenn nicht bereits Seite eins auf dem Bildschirm zu sehen ist, wird "pa" vermindert und "zeile" erhält den Wert eins. Die Ausgabe der jeweiligen Seite, die Berechnung der Zeile "li"

und das Setzen des Cursors werden wiederum dem Hauptprogramm überlassen.

Komplizierter ist die Funktion "Zeile löschen". Gelöscht werden soll die Druckzeile, in der sich der Cursor momentan befindet. Die Zeilennummer ist in "li" vorhanden. Der zu löschende String ist daher "u$(li)". Das Löschen dieses Strings ist jedoch nicht ausreichend, da der nachfolgende Text aufgerückt werden muß.

Dieses Aufrücken bewirkt eine Schleife, in der jeder String ab "u$(zeile+(pa-1)*lpage)", das heißt ab dem Index der aktuellen Druckzeile, bis "lmax-1", das heißt bis zum vorletzten möglichen Index, den Inhalt des nachfolgenden Strings erhält (Zeile 1630-1650). Zum Abschluß wird die letzte Druckzeile mit dem Index "lmax" mit Leerzeichen beschrieben.

Das Einfügen einer neuen Zeile an der momentanen Cursorposition, die nur aus Leerzeichen besteht, verläuft selbstverständlich analog. Wenn sich der Cursor nicht in der letzten möglichen Druckzeile befindet und daher kein Einfügen möglich ist (Zeile 1700), werden alle Druckzeilen ab der aktuellen um eins nach "oben" verschoben, erhalten also höhere Indizes (Zeile 1710-1730). Die Zeile, in der sich der Cursor befindet, wird mit Leerzeichen gefüllt (Zeile 1740).

Das Sortieren von Daten (Zeile 1770-1930) will ich nicht näher erläutern, abtippfertige Sortieralgorithmen gibt es in Hülle und Fülle. Wenn Sie den verwendeten Algorithmus in Ihre eigenen Programme einbauen wollen, so beachten Sie bitte, daß "a" die Anzahl der zu sortierenden Strings (= höchster Index) und "u$(...)" das zu sortierende Stringarray ist. Der eigentliche Sortieralgorithmus beginnt ab Zeile 1860 und endet in Zeile 1920.

Ab Zeile 1950 beginnt der Programmteil "Ausdrucken". Ausgedruckt werden sollen nur Strings, die Daten enthalten, das heißt die nicht ausschließlich aus Leerzeichen bestehen.

In einer Schleife (Zeile 1960-1970) werden - beim letzten beginnend - alle Strings daraufhin untersucht. Nach dem Beenden

der Schleife befindet sich in "i" der Index des - vom Ende des Stringarrays aus gesehen - Index des ersten Strings, der Daten enthält.

Nachdem ein Kanal geöffnet wurde, werden diese Strings in einer weiteren Schleife auf den Drucker ausgegeben, wobei jeweils drei vorausgehende Leerzeichen gesendet werden, um einen Abstand zum linken Papierrand zu erhalten. Nach dem Ausdruck aller Daten wird der Kanal wieder geschlossen.

"Abspeichern eines Textes" (ab Zeile 1960) wird nur dann ausgeführt, wenn zuvor eine Sicherheitsabfrage mit "j" beantwortet wurde. Zuerst wird der String mit dem höchsten Index ermittelt, der Daten enthält (Zeile 2260-2270). Der Benutzer wird nach dem Namen gefragt, unter dem der Text gespeichert werden soll (Zeile 2280).

Nachdem eine Diskettendatei unter diesem Namen zum Lesen geöffnet wurde, wird in diese Datei die Variable "a" geschrieben, die die Anzahl der abzuspeichernden Strings enthält. Die Daten selbst werden erst anschließend auf die Diskette geschrieben. Zum Schluß wird die Datei wieder geschlossen.

"Laden eines Textes" (Zeile 2050-2190) wird ebenfalls nur nach Beantwortung einer Sicherheitsabfrage ausgeführt. Alle Strings werden initialisiert, das heißt mit Leerzeichen gefüllt (Zeile 2100).

Zuerst wird eine eventuell bereits unter dem angegebenen Namen vorhandene Datei gelöscht, anschließend wird eine Datei unter dem jeweiligen Namen zum Schreiben geöffnet und die Variable "a" eingelesen, die die Anzahl der abgespeicherten Strings enthält. Diese werden anschließend Zeichen für Zeichen eingelesen (Zeile 2140-2170) und danach der geöffnete Kanal wieder geschlossen.

Der Programmteil "Suche" beginnt ab Zeile 2370 und endet mit Zeile 2690. Der Bildschirm wird gelöscht, der Benutzer erhält Informationen zur Suche und wird aufgefordert, den (die) Such-

begriff(e) einzugeben (Zeile 2380-2450). Zur Eingabe des Suchbegriffs wird die im Programm vorhandene Eingaberoutine verwendet (Zeile 2460). Angenommen werden nur Eingaben, die mit "Return" beendet wurden (Zeile 2470). Der Suchbegriff befindet sich anschließend in "u$(0)".

In Zeile 2490 werden alle sogenannten "nachlaufenden" Spaces entfernt, die Eingabe um jeweils ein Leerzeichen am rechten Stringende gekürzt, bis zum Schluß die echten Eingabedaten übrigbleiben (bedenken Sie, daß "u$(0)" in Zeile 2460 mit 39 Leerzeichen gefüllt wurde!).

In den Zeilen 2500-2530 wird ermittelt, wie viele Suchbegriffe eingegeben wurden. Die Anzahl befindet sich anschließend in "z", und die Positionen des Sonderzeichens "^", das zur Trennung der Suchbegriffe verwendet wird, in den Variablen "i(1)" bis "i(z)".

Nun folgt die eigentliche Suche: Der erste String wird untersucht. Wenn nicht alle eingegebenen Suchbegriffe in diesem String vorhanden sind, wird der nächste String untersucht. Wurde ein String gefunden, der alle eingegebenen Suchbegriffe enthält, wird er auf dem Bildschirm ausgegeben und der Benutzer gefragt "Weitersuchen (j/n) ?". Antwortet der Benutzer mit "j", geht die Suche weiter. Antwortet der Benutzer jedoch mit "n", ist die Suche beendet und "zeile" (die aktuelle Druckzeile) erhält den Index des gefundenen Strings. Aus "zeile" und "lpages" wird die aktuelle Seite "page" errechnet und zum Hauptprogramm zurückgekehrt.

4.3.2. Vokabeltrainer

Der Vokabeltrainer erlaubt die Abfrage einer einzelnen Vokabel (Sie erhalten die Übersetzung in der betreffenden Sprache), einen Vokabeltest, bei dem Sie die Anzahl der abzufragenden Vokabeln und die Richtung (von Sprache "A" nach "B" oder aber umgekehrt) selbst bestimmen können, und die Ausgabe aller in dem "Wörterbuch" vorhandener Vokabeln.

Programmbedienung

Vor der eigentlichen Programmbenutzung erwartet Sie die arbeitsreiche Aufgabe, einen eigenen Wortschatz zu erstellen. Die Vokabeln werden nicht in einer Datei abgespeichert, sondern befinden sich in Form sogenannter "Data-Zeilen" im Programm selbst.

Sie finden am Programmende mehrere Data-Zeilen, die als Beispiel dienen und mehrere englische Vokabeln und deren deutsche Entsprechung enthalten. Zum Ausprobieren können Sie diese Vokabeln verwenden. Es ist jedoch ebenfalls möglich - statt diese Demonstrationszeilen abzutippen - sofort eigene Vokabeln einzugeben.

Zur praktischen Nutzung des Programms ist es unerläßlich, daß Sie sich Ihr eigenes "Wörterbuch" zusammenstellen, wobei jedoch mehrere Punkte unbedingt beachtet werden müssen. Wichtig ist die Form, in der Sie Ihre Vokabeln eingeben:

(Zeilennummer) data "(Vok.A)","(Vok.B)"

In einer Zeile müssen sich zwei Datas befinden, die Vokabel aus Sprache "A" und - durch ein Komma getrennt - die entsprechende Vokabel aus Sprache "B", zum Beispiel zuerst ein deutsches Wort und sein englisches Pendant:

```
1020 data "kriechen","crawl"
```

Zeile 1020 muß unbedingt die erste für die Vokabeln verwendete Zeilennummer sein! Die für die Vokabeln verwendeten Zeilennummern müssen in Zehnerschritten durchnumeriert sein (zum Beispiel: 1020... 1030... usw.)! Die letzte Data-Zeilen muß aus den beiden Datas "x","x" bestehen (siehe Zeile 1170)!

Wenn Sie einen der genannten Punkte nicht beachten, werden unweigerlich Fehler beim Programmlauf auftreten!

Ich gehe nun davon aus, daß Sie entweder die als Beispiel gedachten Data-Zeilen oder aber Ihre eigenen Vokabeln eingegeben haben. Nach dem Starten des Programms sehen Sie ein Auswahlmenü. Sie können eine der folgenden Funktionen auswählen:

1. Abfrage einer Vokabel
2. Vokabeltest
3. Ausgabe einer Vokabel

Wählen Sie die entsprechende Funktion an, indem Sie eine der zugehörigen Zahlentasten "1", "2" oder "3" betätigen.

Abfrage einer Vokabel:

Sie werden nun aufgefordert, eine Vokabel einzugeben. Ob diese Vokabel der Sprache "A" oder der Sprache "B" angehört, ist unwichtig. In jedem Fall zeigt Ihnen das Programm die zugehörige Entsprechung in der jeweils anderen Sprache.

Sollte die Vokabel in den von Ihnen eingegebenen Data-Zeilen nicht existieren, erhalten Sie die Meldung "Nicht gefunden!!!".

Nach jeder Abfrage werden Sie erneut zur Eingabe einer Vokabel aufgefordert. Abbrechen können Sie diese Funktion durch die Eingabe von "ende" anstelle einer Vokabel. Die Eingabe von "ende" führt zum Menü zurück.

Vokabeltest:

Das Programm fordert Sie auf einzugeben, wie viele Vokabeln abgefragt werden sollen. Die Anzahl ist beliebig.

Wenn Sie die von Ihnen gewünschte Anzahl eingegeben haben, werden Sie gefragt, ob die Abfrage in Richtung von Sprache "A" nach "B" oder umgekehrt verlaufen soll. Die Richtung "A" nach "B" bedeutet, daß Ihnen die ersten Vokabeln der Data-Zeilen vorgegeben werden und Sie das Äquivalent (die zweite Vokabel der betreffenden Data-Zeile) eingeben sollen. Diese Richtung wird durch Betätigen der Taste "1" gewählt.

Wird Taste "2" gedrückt, verläuft die Abfragerichtung umgekehrt. Die zweiten Vokabeln der Data-Zeilen dienen als Vorgaben, zu denen Sie das Äquivalent finden sollen.

Das Programm wählt nun per Zufall eine Vokabel aus und fordert Sie auf, das entsprechende Äquivalent einzugeben. Anschließend erhalten Sie entweder die Meldung "Richtig!!!" oder aber "Falsch!!!". Dieser Vorgang wiederholt sich entsprechend der von Ihnen angegebenen Anzahl an Durchgängen.

Nach Beendigung aller Durchgänge erhalten Sie Auskunft über Ihre Erfolgsquote und können den Test wiederholen oder aber zum Menü zurückkehren.

Ausgabe aller Vokabeln:

Diese Funktion erlaubt das Durchgehen - und damit Lernen - aller vorhandenen Vokabeln. Die erste Vokabel und ihre Übersetzung wird auf dem Bildschirm ausgegeben. Durch Betätigung der Leertaste wird zur nächsten Vokabel "weitergeblättert". Zum Menü kehren Sie zurück, wenn Sie eine beliebige andere Taste drücken.

Programmlisting

```
100 rem -----vokabeltrainer-----
110 rem ---(jan 1986/s.baloui)--
120 :
130 rem ---variablenliste---
140 rem tr=trefferanzahl
```

```
150 rem ad=anzahl der versuche
160 rem r=richtung der abfrage
170 rem av=anzahl vorhandener vokabeln
180 :
190 read a$,b$
200 if a$<>"x" then av=av+1:goto 190:else restore:rem av
ermitteln
210 :
220 rem -----hauptprogramm-----
230 print chr$(14):rem gross/klein
240 scnclr
250 :
260 print tab(9)"Vokabeltrainer"
270 print tab(9)"--------------"
280 print:print
290 print tab(5)"(1) Abfrage einer Vokabel"
300 print tab(5)"(2) Vokabeltest"
310 print tab(5)"(3) Ausgabe aller Vokabeln"
320 print:print
330 print tab(5)"Taste 1, 2 oder 3 druecken"
340 getkey a$:i=instr("123",a$)
350 if i=0 then 340:else on i gosub 390,580,850
360 goto 240
370 :
380 rem ---abfr.einer vokabel---
390 char 0,0,20,"Vokabel"
400 input v$
410 if v$="ende" then return:rem zum menue
420 char 0,0,21,"Entsprechung:"
430 e$="Nicht gefunden!"
440 restore:rem data-zeiger initial.
450 :
460 for i=1 to av
470 read a$,b$:rem 2 vokabeln lesen
480 if instr(a$,v$) then e$=b$:goto 520
490 if instr(b$,v$) then e$=a$:goto 520
500 next
510 :
520 char 0,14,21,e$
530 getkey a$
```

```
540 gosub 960:rem zeilen loeschen
550 goto 390
560 :
570 rem ---vokabeltest---
580 char 0,0,20,"Wie viele Durchgaenge"
590 input ad
600 char 0,0,21,"(1) a-b  oder  (2) b-a ?"
610 getkey a$
620 r=instr("12",a$):if r=0 then 610
630 gosub 960:rem zeilen loeschen
640 tr=0
650 :
660 for i=1 to ad
670 rz=int(rnd(0)*(av)):rem data-zeile ermitteln
680 restore 1020+10*rz:rem data-zeiger setzen
690 read a$,b$:rem 2 datas lesen
700 if r=2 then s$=a$:a$=b$:b$=s$:rem datas vertauschen?
710 char 0,0,20,a$:rem vokabel ausgeben
720 input v$:rem abfrage der entsprechung
730 if instr(b$,v$) then tr=tr+1:e$="richtig!!!":else
e$="Falsch!!!"
740 char 0,0,21,e$
750 getkey a$
760 gosub 960:rem zeilen loeschen
770 next
780 :
790 char 0,0,20,str$(tr)+" Treffer bei"+str$(ad)+"
Versuchen"
800 char 0,0,21,"Weiter (j/n) ?"
810 getkey a$:
820 if a$="j" then gosub 960:goto 580:else return
830 :
840 rem ---vokabeln ausgeben---
850 scnclr
860 restore:rem data-zeiger init.
870 :
880 for i=1 to av
890 read a$,b$:rem 2 datas lesen
900 print a$ tab(20)b$
910 getkey a$
```

```
920 if a$<>" " then 940
930 next
940 return
950 :
960 rem ---zeilen loeschen---
970 char 0,0,20,"                                        ":
rem 40 spaces
980 char 0,0,21,"                                        ":
rem 40 spaces
990 return
1000 :
1010 rem ---vokabeln---
1020 data "versammeln","convene"
1030 data "kriechen","crawl"
1040 data "ziehen","draw"
1050 data "heilig","holy"
1060 data "reenlist","wiederverpflichten"
1070 data "reduplicate","verdoppeln"
1080 data "footlights","Rampenlicht"
1090 data "civil","den Staat betreffend"
1100 data "letter","Buchstabe"
1110 data "rail","Schiene"
1120 data "relapse","zurueckfallen"
1130 data "stand by","zur Seite stehen"
1140 data "timid","furchtsam"
1150 data "washable","waschecht"
1160 data "wriggle","sich winden"
1170 data "x","x":rem endezeichen !!!
```

Programmablauf

Nach dem Starten des Programms muß zuerst ermittelt werden, wie viele Vokabeln vorhanden sind. In Zeile 190 werden zwei Datas eingelesen und die Variable "av" erhöht. Dieser Vorgang wird wiederholt (Zeile 200), bis die Datas "x" und "x" eingelesen werden, die als Endmarkierung verwendet werden. Danach wird der Data-Zeiger zurückgesetzt ("RESTORE"), und in "av" befindet sich die Anzahl der im Programm enthaltenen Vokabeln (inclusive deren Übersetzungen).

Das Auswahlmenü bietet nur eine Besonderheit, die Art und Weise der Sprungverteilung. In Zeile 340 wird auf eine Taste gewartet. Handelt es sich nicht um eine der Tasten "1", "2" oder "3" (wird mit "INSTR" überprüft), wird auf einen weiteren Tastendruck gewartet. Ansonsten wird der von "INSTR" ermittelte Wert zum Aufruf des jeweiligen Unterprogramms verwendet (Zeile 350).

Abfrage einer Vokabel:

Die Ausgabe der Übersetzung einer Vokabel wird von dem Unterprogramm "Abfr.einer Vokabel" vorgenommen (Zeilen 380-550). Die vom Benutzer eingegebene Vokabel wird der Variablen "v$" zugewiesen. Wurde "ende" eingegeben, folgt die Rückkehr zum Auswahlmenü (Zeile 410).

Nachdem der Data-Zeiger vor Beginn der Suche zurückgesetzt wurde, werden alle Data-Zeilen (eins bis "av") nach der eingegebenen Vokabel durchsucht.

Je zwei Vokabeln werden eingelesen und mit der eingegebenen verglichen (Zeile 460-500). Fällt ein Vergleich positiv aus, wird "e$" die zweite in der betreffenden Data-Zeile vorhandene Vokabel zugewiesen, das heißt die Übersetzung (Zeile 480-490), und die Schleife verlassen.

"e$" wird nun ausgegeben und enthält entweder die gesuchte Übersetzung oder aber die Meldung "Nicht gefunden!!!", mit der "e$" vor Beginn der Suche initialisiert wurde (Zeile 430).

Es folgt ein Sprung zum Anfang des Unterprogramms, das daher in einer Endlosschleife läuft, die nur durch die Eingabe von "ende" verlassen werden kann.

Vokabeltest:

Nachdem die Anzahl der Durchgänge "ad" und die Abfragerichtung "r" erfragt wurde (Zeile 580-620), beginnt eine Schleife, deren Endwert von der Anzahl an Durchgängen abhängt.

In dieser Schleife wird zuerst eine Zufallszahl zwischen null und "av-1" ermittelt (Zeile 670). Diese Zufallszahl wird zur Ermittlung einer beliebigen Data-Zeile eingesetzt (Zeile 680), auf die der Data-Zeiger mit dem Befehl "RESTORE N" direkt gesetzt wird.

Ein Beispiel: Die Zufallszahl null wird ermittelt. Die Formel "1020+10*rz" ergibt die Zeile 1020, das heißt die erste Data-Zeile. Hat die Zufallszahl "rz" dagegen den Wert fünf, wird der Data-Zeiger auf Zeile 1070 gesetzt, also auf die sechste Data-Zeile.

Diese Methode führt jedoch zu Fehlern, wenn die erste Data-Zeile nicht die Zeilennummer 1020 besitzt, der Zeilenabstand nicht konstant zehn beträgt, oder in einer Data-Zeile nicht jeweils eine Vokabel und deren Übersetzung vorhanden sind bei der Programmbedienung. Deshalb wurde eindeutig vorgeschrieben, auf welche Weise ein Wortschatz einzugeben ist.

Die beiden Datas der ausgewählten Zeile werden nun in "a$" und "b$" eingelesen (Zeile 690). Da die in "a$" eingelesene Vokabel die Vorgabe für den Benutzer darstellen soll (siehe Zeile 710), werden beide Strings miteinander vertauscht, wenn die Abfragerichtung von der zweiten zur ersten Vokabel einer Data-Zeile verlaufen soll (Zeile 700).

Die vom Benutzer eingegebene Antwort "v$" (Zeile 720) wird mit der gesuchten Übersetzung "b$" verglichen (Zeile 730), die Variable "tr", die die Anzahl richtiger Versuche enthält, erhöht, wenn beide identisch sind, und dem Benutzer die Meldung "Richtig!!!" beziehungsweise "Falsch!!!" ausgegeben (Zeile 730-740).

Wenn der Benutzer eine beliebige Taste drückt, wird die nächste Vokabel abgefragt (Zeile 750-770). Nach "ad" Durchgängen wird das Resultat ausgegeben, und der Benutzer gefragt, ob er einen weiteren Vokabeltest durchführen will (Zeile 790-820).

Ausgabe aller Vokabeln:

Zur Ausgabe aller Vokabeln (Zeile 840-940) wird der Data-Zeiger zurückgesetzt (Zeile 860), anschließend die erste Vokabel inclusive der Übersetzung gelesen und ausgegeben (Zeile 890-900), auf die Betätigung der Leertaste gewartet und die nächste Vokabel gelesen.

Die Schleife, in der dieser Vorgang abläuft, wird vorzeitig verlassen, wenn der Benutzer eine andere als die Leertaste drückt (Zeile 920).

4.3.3. KFZ-Überwachung

Das folgende Programm bietet zwei sehr nützliche Funktionen: Zum einen ermöglicht es Ihnen auf problemlose Art und Weise die Errechnung des Durchschnittsverbrauchs Ihres Autos, Motorrades oder sonstigen Kraftfahrzeugs. Zum anderen erinnert es Sie an die Einhaltung der regelmäßigen Inspektionstermine. Das Programm benötigt zur Erfüllung dieser Aufgaben die dauerhafte Abspeicherung wichtiger Daten. Hierzu wird eine Datei auf der Diskette verwaltet, die jeweils nach dem Programmstart geladen und zum Abschluß der Arbeiten mit dem Programm - mit den neuesten Werten versehen - wieder abgespeichert wird.

Programmbedienung

Nach dem Starten stellt Ihnen das Programm die Frage "Erstbenutzung (j/n) ?". Wenn Sie das Programm zum ersten Mal benutzen, beantworten Sie die Frage bitte durch Betätigung der Taste "j". Antworten Sie bei jeder weiteren Benutzung mit "n".

Die folgenden Fragen werden Ihnen nur dann gestellt, wenn Sie das Programm zum ersten Mal benutzen:

"Anfangskilometerstand?": Geben Sie bitte jenen Kilometerstand ein, den Ihr Tachometer momentan anzeigt.

"Inspektionsintervall in km?": Die Antwort können Sie der Bedienungsanleitung Ihres Kraftfahrzeugs entnehmen. Üblich sind Inspektionsintervalle von zum Beispiel 5000 km, wobei abwechselnd sogenannte "kleine" und "große" Inspektionen durchgeführt werden.

"Letzte Inspektion bei km?": Geben Sie auf diese letzte Frage bitte den Kilometerstand ein, der vorlag, als Sie zuletzt eine Inspektion durchführen ließen.

Diese drei Fragen werden Ihnen nur bei der Erstbenutzung des Programms gestellt. Bei jeder weiteren Benutzung werden ständig benötigte Daten von der Diskette gelesen. Diese Daten werden unter einem von Ihnen gewählten Namen in eine Datei geschrieben, die nach dem Starten des Programms geladen wird. Bei jeder außer der erstmaligen Benutzung des Programms werden Sie daher nach dem Namen gefragt, den Sie der Datei gaben, und diese anschließend geladen.

Auf dem Bildschirm erscheint nun ein Auswahlmenü. Sie können eine der folgenden Funktionen anwählen:

1. Verbrauch/Inspektion
2. Getankt
3. Inspektion durchgeführt
4. Ende der Arbeit

Zur Anwahl einer dieser Funktionen betätigen Sie bitte eine der Zahlentasten eins bis vier. Nach Durchführung einer Funktion (Ausnahme "Ende der Arbeit") kehren Sie immer zum Auswahlmenü zurück. Beenden Sie das Programm bitte immer mit der

dafür vorgesehenen Funktion des Menüs und nicht durch einen Programmabbruch (zum Beispiel durch Drücken von "STOP").

Verbrauch/Inspektion:

Das Programm will von Ihnen nach Anwahl dieser Funktion den momentanen Kilometerstand wissen. Nach Eingabe der jeweiligen Zahl erfahren Sie:

a) Den bisherigen Durchschnittsverbrauch
b) Wann die nächste Inspektion durchgeführt werden muß

Getankt:

Wählen Sie diese Funktion immer an, wenn Sie getankt haben. Das Programm benötigt zur Ermittlung Ihres Durchschnittsverbrauchs unbedingt die kompletten Daten über die verbrauchte Literzahl und den Kilometerstand beim Tanken.

Diese beiden Informationen, bei welchem Kilometerstand und wieviel Liter Sie getankt haben, werden nun von Ihnen erfragt. Anschließend erhalten Sie Auskunft über den bisherigen Durchschnittsverbrauch (in l/100 km).

Inspektion durchgeführt:

Nach Durchführung einer Inspektion sollten Sie das Programm durch Anwahl dieser Funktion davon unterrichten. Es will von Ihnen erfahren, bei welchem Kilometerstand die Inspektion durchgeführt wurde und welcher Kilometerstand momentan von Ihrem Tachometer abzulesen ist. Da das Programm bei der Erstbenutzung über das Inspektionsintervall unterrichtet wurde, kann es nun errechnen, wieviel Kilometern noch bis zur nächsten Inspektion verbleiben und Ihnen diesen Wert ausgeben.

Ende der Arbeit:

"Ende der Arbeit" ist die einzige Funktion, die keinerlei weitere Angaben von Ihnen benötigt, sondern sofort ausgeführt wird. Diese Funktion muß unbedingt angewählt werden, wenn Sie das Programm verlassen wollen. Sie dürfen keinesfalls die "STOP"-Taste drücken oder den Rechner einfach ausschalten.

Wurde eine der beiden Funktionen "Getankt" beziehungsweise "Inspektion durchgeführt" angewählt, werden die von Ihnen eingegebenen Daten vor Beenden der Arbeit auf die Diskette geschrieben. Der Datei, in der die Daten gespeichert werden, können Sie einen beliebigen Namen geben, den das Programm zuvor von Ihnen wissen will.

Hatten Sie das Programm nur geladen und gestartet, um Auskunft über den bisherigen Durchschnittsverbrauch und den nächsten Inspektionstermin zu erhalten, müssen keine Daten abgespeichert werden und das Pogramm wird beendet.

Programmlisting

```
100 rem -bezinverbrauch/inspektionen-
110 rem ----(januar 1986/s.baloui)---
120 :
130 rem ---variablen---
140 rem cs=cursorspalte
150 rem cz=cursorzeile
160 rem cr=cursor-setz-routine
170 rem ef$=erstbenutzungsflag (j/n)
180 rem af=flag: aenderungen vorgenommen (0/1)
190 rem ak=anfangskilometerstand
200 rem dv=bisher. durchschn.-verbrauch
210 rem sl=summe verbrauchter liter
220 rem li=letzte inspektion (km)
230 rem ii=inspektionsintervall in km
240 rem mk=momentaner kilometerstand
250 rem al=anzahl getankter liter
```

```
260 :
270 cs=202
280 cz=205
290 cr=55464
300 ho$=chr$(19):rem steuerzeichen 'cursor home'
310 :
320 rem -----hauptprogramm-----
330 print chr$(14):rem gross/klein
340 scnclr
350 print "Erstbenutzung des Programms (j/n) ?"
360 getkey ef$
370 if ef$="n" then gosub 960:goto 430:rem datei laden
380 :
390 print:input "Anfangskilometerstand";ak
400 input "Inspektionsintervall in km";ii
410 input "Letzte Inspektion bei km";li
420 :
430 scnclr
440 print ho$;
450 print "   Verbrauch + Inspektionsintervalle"
460 print "   ---------------------------------"
470 print:print
480 print "   (1) Verbrauch/Inspektion"
490 print "   (2) Getankt"
500 print "   (3) Inspektion durchgefuehrt"
510 print "   (4) Ende der Arbeit"
520 char 0,3,10,"Zahl zwischen 1 und 4 eingeben"
530 getkey a$:nr=val(a$):if nr<1 or nr>4 then 530
540 if nr<>4 then on nr gosub 590,660,760:goto 520
550 if ef$="j" or af then gosub 840:rem abspeichern?
560 end
570 :
580 rem ---verbrauch/inspektion---
590 gosub 1080
600 input "Momentaner Kilom.-Stand";mk
610 print "Bisheriger Durchschn.-Verbrauch:"dv"l"
620 print "Naechste Inspektion in"ii-(mk-li)"km"
630 return
640 :
650 rem ---getankt---
```

```
660 af=1
670 gosub 1080
680 input "Getankt bei Kilom.-Stand";mk
690 input "Wieviel Liter";al
700 sl=sl+al:rem summe liter um al erhoehen
710 dv=sl/((mk-ak)/100):rem durchschn.verbrauch
720 print "Durchschnittsverbrauch:"dv"l"
730 return
740 :
750 rem ---inspektion durchgefuehrt---
760 af=1:rem aenderungsflag setzen
770 gosub 1080
780 input "Inspektion bei Kilom.-Stand";li
790 input "Momentaner Kilom.-Stand";mk
800 print "Naechste Inspektion in"ii-(mk-li)"km"
810 return
820 :
830 rem ---datei abspeichern---
840 gosub 1080
850 input "Name der Datei";n$
860 open 15,8,15,"s:"+n$:close 15:open 1,8,2,n$+",s,w"
870 print#1,ak
880 print#1,sl
890 print#1,dv
900 print#1,li
910 print#1,ii
920 close1
930 return
940 :
950 rem ---datei laden---
960 gosub 1080
970 input "Name der Datei";n$
980 open 1,8,2,n$+",s,r"
990 input#1,ak
1000 input#1,sl
1010 input#1,dv
1020 input#1,li
1030 input#1,ii
1040 close1
1050 return
```

```
1060 :
1070 rem ---init ausgaben---
1080 poke cs,0:poke cz,10:sys cr:rem cursor setzen
1090 for i=1 to 12
1100 print "                                      "
1110 next
1120 poke cs,0:poke cz,17:sys cr:rem cursor setzen
1130 return
```

Programmablauf

Folgende Daten müssen ständig für das Programm verfügbar sein:

1. Der Anfangskilometerstand "ak"
2. Die Summe der bisher getankten Liter "sl"
3. Der bisherige Durchschnittsverbrauch "dv"
4. Der Kilometerstand bei der letzten Inspektion "li"
5. Das Inspektionsintervall "ii"

Diese Daten werden auf Band in eine Datei geschrieben. Die Datei wird vor Beginn der Arbeit mit dem Programm geladen und - falls sich die Daten durch die Programmbenutzung änderten - nach Beenden der Arbeit abgespeichert.

Da bei der Erstbenutzung des Programms diese Datei noch nicht existiert und daher auch nicht geladen werden kann, stellt dieser Fall eine Ausnahme dar, die besonders behandelt wird: Es wird keine Datei geladen, dafür werden vom Benutzer mehrere Daten erfragt, die bei jeder weiteren Programmbenutzung benötigt werden (Zeilen 390-410).

Verbrauch/Inspektion:

Zuerst wird ein Unterprogramm angesprungen (Zeile 590), dessen einzige Aufgabe es ist, die untersten Zeilen des Bildschirms zu löschen, die zur Informationsausgabe benutzt werden.

Nach Eingabe des momentanen Kilometerstandes "mk" durch den Benutzer wird der bisherige Durchschnittsverbrauch "dv" und die Kilometerzahl, nach der die nächste Inspektion erfolgt, ausgegeben. Diese Zahl wird ermittelt, indem die Differenz zwischen momentanem Kilometerstand und Kilometerstand bei der letzten Inspektion errechnet wird, und diese Differenz vom Inspektionsintervall subtrahiert wird (Zeile 620).

Getankt:

Nach Eingabe des Kilometerstandes, bei dem getankt wurde, "mk") und der Literzahl "al" wird die Gesamtsumme getankter Liter "sl" um "al" erhöht und der Durchschnittsverbrauch ermittelt. Der Verbrauch wird errechnet, indem die Litersumme durch die - durch 100 geteilte - Anzahl gefahrener Kilometer "mk-ak" geteilt wird (Zeile 710).

Inspektion durchgeführt:

Die Errechnung der Anzahl an Kilometern, nach denen die nächste Inspektion durchgeführt werden muß, verläuft analog zur Ermittlung des gleichen Wertes im Rahmen der Funktion "Verbrauch/Inspektion": Der Kilometerstand "li" bei der zuletzt durchgeführten Inspektion und der momentane Kilometerstand "mk" werden erfragt. Durch Subtraktion der Differenz zwischen "mk" und "li" von "ii", dem Inspektionsintervall, ergibt sich der gesuchte Wert.

Ende der Arbeit:

Wenn das Änderungsflag "af" den Wert eins besitzt oder aber eine Programmerstbenutzung vorliegt ('ef$="j"'), wird nach Anwahl dieser Funktion die Datei abgespeichert, ansonsten wird das Programm sofort beendet (Zeile 550). "af" hat nach dem Programmstart den Wert null. Den Wert eins erhält es, wenn die Funktion "Getankt" oder die Funktion "Inspektion durchgeführt" angewählt wurde (Zeilen 660 und 760). In beiden Fällen werden Daten, die in der Datei abgespeichert sind, durch Eingaben des Benutzers geändert ("sl" bzw. "li").

Zum Abspeichern der Datei wird der Benutzer zuerst nach dem Namen gefragt, den er der Datei geben will (Zeile 850). Nachdem eine eventuell bereits unter diesem Namen vorhandene Datei gelöscht wurde, wird eine Diskettendatei zum Schreiben geöffnet (Zeile 860) und die genannten Daten auf die Diskette geschrieben (Zeilen 870-910), und zum Schluß wird die Datei geschlossen (Zeile 920). Das Abspeichern der geänderten Daten ist damit beendet. Das Einlesen der Datei bei jeder außer der ersten Programmbenutzung verläuft analog (Zeilen 950-1050).

Ausgabezeilen löschen:

Im Unterprogramm zum Löschen der Bildschirmzeilen, die zur Ausgabe verwendet werden (Zeilen 1070-1130), sind vor allem zwei Zeilen interessant, in denen der Cursor direkt auf eine bestimmte Bildschirmposition gesetzt wird.

In Zeile 1080 soll der Cursor auf Spalte null von Zeile zehn gesetzt werden. Diese beiden Werte werden in die Speicherzellen "gepokt", in denen der PLUS/4 die Informationen über die aktuelle Spalte und Zeile des Cursors speichert. Durch den Aufruf von "cr" (cr=55464) wird der Cursor auf die angegebene Position gesetzt. Auf die gleiche Weise wird in Zeile 1120 der Cursor auf Spalte null von Zeile 17 gesetzt.

Folgende Schritte sind zum direkten Setzen des Cursors nötig:

1. `poke 205,(Spalte)`
2. `poke 202,(Zeile)`
3. `sys 55464`

4.4. Mathematik

Der Ablauf der in diesem Kapitel vorgestellten Programme "Determinantenberechnung" und "Zahlentheorie" wird von mir nicht erläutert werden. Beide Programme wurden freundlicherweise von V.Bühn zur Verfügung gestellt und von mir in eine etwas strukturierte Form gebracht. Mathematisch Interessierte dürften jedoch keine Schwierigkeiten haben, den Programmablauf anhand der Listings nachzuvollziehen.

4.4.1. Zinzeszins

"Zinzeszins" ist ein sehr einfaches Programm, das jedoch mühsame Rechenarbeit ersparen helfen kann. Es berechnet das Endkapital, das sich aus einem bestimmten Anfangsbetrag bei einer festen Verzinsung innerhalb einer beliebigen Laufzeit ergibt. Voraussetzung ist jedoch, daß auch die Zinserträge wieder angelegt werden.

Programmbedienung

Das Programm benötigt zur Berechnung Informationen über den Zinssatz, die Laufzeit und das Anlagekapital. Diese Daten werden nach dem Programmstart erfragt. Anschließend wird auf dem Bildschirm sofort das berechnete Endkapital ausgegeben.

Programmlisting

```
100 rem -----zinzeszinsberechnung-----
110 rem ----(januar 1986/s.baloui)----
120 :
130 rem ---variablen---
140 rem zs=zinssatz
150 rem lz=laufzeit
160 rem ak=anlagekapital
170 rem ek=endkapital
180 :
190 printchr$(14):rem gross/klein-schrift
200 scnclr
210 input"Zinssatz (in Prozent)";zs
220 input"Laufzeit (in Jahren)";lz
230 input"Anlagekapital";ak
240 :
250 ek=ak
260 for i=1 to lz
270 ek=ek+ek/100*zs
280 next
290 :
300 print:print
310 printak"DM mit"zs"Prozent ueber"lz"Jahre"
320 print" angelegt ergibt als Endkapital"ek"DM
```

Programmablauf

Der Programmablauf ist problemlos zu verstehen: Nachdem vom Benutzer der Zinssatz "zs", die Laufzeit "lz" und das Anlagekapital "ak" eingegeben wurden (Zeilen 210-230), wird das Endkapital mit Hilfe einer Schleife berechnet. Diese Schleife läuft vom Startwert eins bis zum Endwert "lz", der jeweiligen Laufzeit in Jahren. Bei jedem Schleifendurchgang wird der Zinsbetrag berechnet, der sich aus dem vorhandenen Kapital in einem Jahr ergibt und dieses Kapital um die Zinsen erhöht (Zeile 270). Nach Beendigung aller Schleifendurchgänge wird das ermittelte Endkapital auf dem Bildschirm ausgegeben.

4.4.2. Determinantenberechnung

Dieses Programm berechnet die Determinante eines linearen Gleichungssystem, das aus maximal 24 Gleichungen bestehen kann.

Programmbedienung

Nach dem Programmstart müssen Sie die Anzahl "n" der Reihen angeben, aus denen das betreffende Gleichungssystem besteht. Anschließend geben Sie bitte die Parameter jeder Reihe (das heißt die Spaltenwerte "a(i,1)" bis "a(i,n)") nacheinander ein.

Wenn die Parameter aller Reihen des Gleichungssystems vollständig eingegeben wurden, wird die Determinante berechnet und ausgegeben.

Programmlisting

```
100 rem -----determinante-----
110 rem ----(1985/v.buehn)----
120 :
130 scnclr
140 input"wieviele reihen";n
150 n=n-1
160 dim a(n,n),p(n)
170 p=1
180 :
190 for i=0 to n
200 for j=0 to n
210 print "a("i+1;j+1")";
220 input a(i,j)
230 next j,i
240 if n=1 then 370
250 :
260 for k=0 to n-2
270 for l=k to n
```

```
280 if a(k,l)=0 then next l
290 p(k)=a(k,l)
300 for i=0 to n
310 if i=k then next i
320 for j=0 to n
330 if j=k then next j
340 a(i,j)=p(k)*a(i,j)-a(i,k)*a(k,j)
350 next j,i,k
360 :
370 a=a(n-1,n-1)*a(n,n)-a(n-1,n)*a(n,n-1)
380 :
390 for m=0to n-2
400 p=p(m)^(n-1-m)*p
410 next
420 :
430 print a/p
440 :
450 :
460 rem erklaerung:
470 rem berechnung der det. durch
480 rem verdichtung nach chio
490 rem (s. aitken, s. 49)
500 rem in 100-120 wird der pivot gesucht.
510 rem danach werden die zeilen jeweils neu berechnet, nur
die zeile, wo der
520 rem pivot vorkommt, nicht.
530 rem dies bis zu einer 2*2-det.,
540 rem die qua.kreuzprodukt berechnet wird. schliesslich
werden die pivots
550 rem beruecksichtigt und durch den wert von a geteilt.
560 rem max. reihen: 24
```

4.4.3. Zahlentheorie

Ebenso wie das Programm zur Determinantenberechnung stammt auch die "Zahlentheorie" von V.Bühn. Der Programmablauf wird von mir nicht weiter kommentiert, da es den Autor sicherlich verärgern würde, wenn sein Programm eventuell unkorrekt erläutert würde.

"Zahlentheorie" kann ermitteln, ob eine beliebige Zahl eine Primzahl ist, die Primzahlen in einem beliebigen Zahlenintervall berechnen, beliebige Zahlen in ihre Primfaktoren zerlegen, den größten gemeinsamen Teiler (GGT) und das kleinste gemeinsame Vielfache (KGV) einer Zahl ermitteln sowie den Binomialkoeffizienten "n" über "k" berechnen.

Programmbedienung

Nach dem Starten erscheint ein Auswahlmenü. Sie können eine der fünf genannten Funktionen anwählen, indem Sie eine der Zahlentasten eins bis fünf betätigen.

Primzahl:

Geben Sie die betreffende Zahl ein. Ihnen wird mitgeteilt, ob es sich um eine Primzahl handelt. Ist dies nicht der Fall, werden die Primfaktoren ausgegeben, in die die Zahl zerlegt werden kann.

Primzahlerzeugung:

Nachdem Sie Anfang und Ende des gewünschten Intervalls angegeben haben (zum Beispiel: 100-1000) werden auf dem Bildschirm alle in diesem Intervall vorhandenen Primzahlen ausgegeben.

Faktorenzerlegung:

Nach Eingabe einer beliebigen Zahl wird diese in ihre Primfaktoren zerlegt beziehungsweise Ihnen wird mitgeteilt, daß eine solche Zerlegung nicht möglich ist, da die eingegebene Zahl selbst eine Primzahl ist.

GGT und KGV:

Geben Sie auf Aufforderung zwei beliebige Zahlen ein. Auf dem Bildschirm werden der größte gemeinsame Teiler und das kleinste gemeinsame Vielfache der beiden Zahlen ausgegeben.

Binomialkoeffizient:

Nach Eingabe von "n" und "k" ermittelt dieser Programmteil den Binomialkoeffizienten "n" über "k".

Nach Durchführung einer der fünf Programmteile werden Sie aufgefordert, eine beliebige Taste zu drücken. Wenn Sie der Aufforderung nachgekommen sind, wird das Programm neu gestartet und es erscheint wieder das Auswahlmenü.

Programmlisting

```
100 rem -----zahlentheorie-----
110 rem ----(1985/v.buehn)-----
120 :
130 rem ---menue---
140 scnclr
150 print "(1)  primzahl"
160 print "(2)  primzahlerzeugung"
170 print "(3)  faktorzerlegung"
180 print "(4)  ggt und kgv"
190 print "(5)  binomialkoeefizient"
200 getkey a$
210 scnclr
220 on val(a$) goto 250,370,530,660,820
230 goto 140
240 :
250 rem ---primzahl?---
260 print "es wird angezeigt, ob eine zahl prim-":print
"zahl ist oder nicht"
270 print:input "zahl eingeben";n
```

```
280 p=int(n/2)
290 if n/2=p then 330
300 for i=3 to p step 2
310 q=n/i
320 if q<>int(q) then next:print:print"ist primzahl":goto
930
330 print:print "ist keine primzahl":print "zerlegung
lautet:":print
340 i=1
350 goto 570
360 :
370 rem ---primzahlgenerierung---
380 print "die primzahlen von einer anfangszahl"
390 print "bis zu einer endzahl werden angezeigt":print
400 input "anfang";a
410 input "ende";n
420 print
430 if a<2 then print " 2 ";:a=3
440 for j=a to n
450 for i=2 to int(j/2)
460 p=j/i
470 if p<>int(p) then nexti:print j;:goto 500
480 i=int(j/2)
490 next i
500 next j
510 goto 930
520 :
530 rem ---faktorzerlegung---
540 print "eine zahl wird in ihre primfaktoren     zerlegt"
550 print:input "zahl eingeben";n:print
560 i=1
570 i=i+1
580 p=n/i
590 if p<>int(p) then 570
600 print i;
610 i=1
620 n=p
630 if n>=2 then 570
640 goto 930
650 :
```

```
660 rem ---ggt und kgv---
670 print "der groesste gemeinsame teiler und"
680 print "das kleinste gemeinsame vielfache"
690 print "zweier zahlen werden berechnet":print
700 input "1. zahl";a
710 a1=a
720 input "2. zahl";b
730 b1=b
740 if a>b then 750:a=z:a=b:b=z:if a=b then b=1:goto 780
750 q=int(a/b)
760 q=a-q*b
770 if q<>0 then a=b:b=q:goto 750
780 print:print:print "der ggt ist"b
790 print "das kgv ist"a1*b1/b:goto 930
800 goto 930
810 :
820 rem ---binomialkoeffizient---
830 print "berechnet wird n ueber k"
840 input "n";n
850 input "k";k
860 p=1
870 if k=0 then 830
880 for i=n-k+1 to n
890 p=i/(n+1-i)*p
900 next
910 print:print n"ueber"k"ist"p
920 :
930 print:print "q taste druecken!"
940 getkey t$
950 run
```

4.5. Hilfsprogramme

Die im folgenden vorgestellten Hilfsprogramme sind keine eigenständigen Anwenderprogramme. Sie werden eingesetzt, um die Möglichkeiten des PLUS/4 besser auszunutzen, oder aber nicht vorhandene Funktionen zu bieten wie zum Beispiel das "Mergen" von Programmen.

Mehrere der vorgestellten Programme sind Unterprogramme, die Sie in eigene Programme einbauen können. Beachten Sie in diesen Fällen bitte unbedingt das genaue Format des Aufrufs, die Zeilennumerierung und die verwendeten Variablen. Sie sollten vermeiden, in Ihrem Hauptprogramm die gleichen Variablen wie die betreffende Hilfsroutine zu verwenden, um ein versehentliches Ändern dieser Variablen zu vermeiden.

4.5.1. Hardcopy

Mit diesem Hilfsprogramm ist es möglich, den Inhalt des Textbildschirms (nicht des Graphikbildschirms!) auf den gängigen Commodore-Druckern auszugeben.

Programmbedienung

Die eigentliche Hardcopy-Routine beginnt ab Zeile 170 und kann von Ihnen in jedes Programm eingebaut werden. Sollte es Überschneidungen zwischen den von mir verwendeten Zeilennumern (170-280) und den in Ihren Programmen verwendeten Zeilennummern geben, ist eine Umnumerierung der Routine jederzeit möglich. Sie können die Routine vor oder nach den Befehlen Ihres eigenen Programms einbauen. Der Aufruf der Routine erfolgt mit einem "GOSUB"-Befehl zur ersten Zeilennummer (Zeile 170). Wenn der Drucker eingeschaltet ist, wird der momentane Bildschirminhalt ausgedruckt und anschließend erfolgt der Rücksprung in das aufrufende Programm.

Programmlisting

```
100 rem -----bildschirm-hardcopy-----
110 rem ---(januar 1986/s.baloui)----
120 :
130 gosub 180:rem aufruf der routine
140 end
150 :
160 :
```

```
170 rem ---hardcopy-routine---
180 sp$="                                        ":rem 40
spaces
190 open 4,4:open 3,3:rem kleinschrift mit 'open 4,4,7'
200 print chr$(19);
210 for i=1 to 25
220 for ii=1 to 40
230 get#3,a$:mid$(sp$,ii)=a$
240 next ii
250 print#4,sp$
260 next i
270 close 3:close 4
280 return
```

Programmablauf

Das Programm ist zwar sehr kurz, aber dennoch nicht uninteressant. Es gibt viele Möglichkeiten, mit BASIC eine Hardcopy-Routine zu programmieren, ein Problem stellt jedoch immer die für den Ausdruck des kompletten Bildschirms benötigte Zeit dar.

Ich glaube, daß dieses Unterprogramm die schnellste in BASIC denkbare Hardcopy-Routine darstellt. Das Prinzip besteht darin, den Bildschirm als Eingabegerät zu definieren (Standardeingabegerät: Tastatur), und mit dem "GET#"-Befehl von diesem Eingabegerät Zeichen für Zeichen zu lesen, ebenso wie von einer Datei, die sich auf der Cassette befindet.

Zu Beginn der Routine wird der String "sp$" angelegt, der 40 Leerzeichen enthält (Zeile 180). Anschließend werden in Zeile 190 zwei Dateien geöffnet. Eine Datei mit dem Bildschirm als Eingabegerät (Gerätenummer drei), und eine weitere Datei mit dem Drucker als Ausgabegerät (Gerätenummer vier).

In Zeile 200 wird der Cursor auf die linke obere Bildschirmecke gesetzt (chr$(19)=Steuerzeichen für "Cursor home").

Für die eigentliche Hardcopy-Routine werden zwei ineinandergeschachtelte Schleifen benötigt. Die äußere Schleife läuft vom Startwert eins bis zum Endwert 25 (25 Bildschirmzeilen) und die innere Schleife von eins bis 40 (40 Spalten pro Zeile). Innerhalb dieser Schleifen wird mit "GET#" Zeichen für Zeichen vom Bildschirm gelesen und in "sp$" an die Position "ii" übertragen, die der jeweils bearbeiteten Bildschirmspalte entspricht.

Wenn die innere Schleife nach 40 Durchgängen verlassen wird, wurde genau eine Bildschirmzeile abgearbeitet, deren Inhalt sich nun in "sp$" befindet. "sp$" wird auf dem Drucker ausgegeben (Zeile 250), und die Abarbeitung der nächsten Zeile beginnt (Zeile 260).

Wenn der komplette Bildschirminhalt ausgedruckt wurde, werden beide Dateien geschlossen und es erfolgt der Rücksprung aus der Routine.

4.5.2. REM-Killer

Der PLUS/4 stellt einen recht großen Anwenderspeicher zur Ablage von Programmen und Daten zur Verfügung. Vor allem bei Programmen, die extrem große Datenmengen verwalten müssen (Textverarbeitung/Dateiverwaltung), können dennoch Konflikte mit dem verfügbaren Speicherplatz auftreten.

Diese Beschränkungen führen oftmals zu Programmen, in denen auf jegliche Strukturierung und Kommentar-Zeilen verzichtet wird, um Speicherplatz zu sparen. Den Nachteil dieser sogenannten "Spaghetti-Programme" erkennt jeder Programmierer spätestens dann, wenn er nach längerer Zeit Änderungen in seinem Programm vornehmen will und feststellen muß, daß er sein eigenes Programm nicht mehr versteht.

Der REM-Killer ermöglicht es, mit zwei Programmversionen zu arbeiten: Einer Version, die übersichtlich, strukturiert und kommentiert ist, und einer zweiten, weitaus kürzeren, in der mit diesem Hilfsprogramm alle Kommentare gelöscht wurden. Der REM-Killer löscht alle reinen Kommentarzeilen (zum Beispiel

"100 rem ---hauptprogramm---") und alle Zeilen, die nur aus einem Doppelpunkt bestehen und allein zur optischen Strukturierung eingesetzt wurden (zum Beispiel "1000 :").

Sie können daher problemlos gut kommentierte Programme schreiben, die viel Speicherplatz beanspruchen. Wenn diese Programme zu lang werden, das heißt nicht mehr ausreichend Platz für die beim Programmlauf angelegten Variablen vorhanden ist und Sie die Fehlermeldung "out of memory error" erhalten, lassen Sie einfach den REM-Killer auf das Programm los.

Er erzeugt Ihnen ein zweites Programm, das in der Funktion mit dem Originalprogramm identisch, jedoch weitaus kürzer ist.

Sie arbeiten mit dem kürzern Programm, haben jedoch immer noch die kommentierte Version, die verständlich ist, und in der Sie problemlos Programmänderungen vornehmen können. Sollten derartige nachträgliche Änderungen im Originalprogramm vorgenommen worden sein, bearbeiten Sie dieses ebenfalls anschließend wieder mit dem REM-Killer.

Der größte Vorteil des REM-Killers besteht darin, daß der Programmablauf beschleunigt wird, da jede REM-Zeile, die der BASIC-Interpreter bearbeiten muß, den Programmablauf bremst. Der Geschwindigkeitsvorteil ist von Programm zu Programm unterschiedlich, kann jedoch durchaus bis zu etwa 30% betragen.

Vor der Bedienungsanleitung noch ein wichtiger Hinweis: Da der REM-Killer Kommentar- und Doppelpunktzeilen löscht, stehen diese nicht mehr als Sprungziele zur Verfügung!

Schreiben Sie Programme, die Sie eventuell später mit dem REM-Killer bearbeiten wollen, daher möglichst so, daß kein "GOTO"- oder "GOSUB"-Sprung zu einer solchen Zeile führt, da Sie ansonsten einen "undefined statement error" erhalten, wenn die anzuspringende Zeile vom BASIC-Interpreter nicht mehr gefunden wird.

Original	Nach der Bearbeitung
100 gosub 200	100 gosub 200
110 :	120 ....
120	130 ....
130	210 for i=1 to 10
200 rem ---unterprogramm---	220 ....
210 for i=1 to 10	230 ....
220	
230	

Zweitens müssen Sie unbedingt darauf achten, daß sich die Zeilennummern des REM-Killers nicht mit denen des zu bearbeitenden Programms überschneiden.

Da die höchste Zeilennummer des REM-Killers die Zeile Nummer 32 ist, darf die kleinste Zeilennumer des zu bearbeitenden Programms keinesfalls kleiner als 33 sein!

Um welchen Faktor Programme durch den Rem-Killer gekürzt werden, hängt vom Umfang der Kommentare ab. Die ebenfalls in diesem Buch vorgestellte Textverarbeitung zum Beispiel wird um circa 30% kürzer. Diese Komprimierung der Programme hat den zusätzlichen Vorteil, daß die Ladezeiten mit dem Diskettenlaufwerk ebenfalls kürzer werden. Ein dritter Vorteil besteht wie erwähnt darin, daß der BASIC-Interpreter des PLUS/4 durch das Entfallen der Kommentarzeilen weniger Arbeit hat, was dazu führt, daß die gekürzten Programme zum Teil erheblich schneller ablaufen (vor allem die bereits erwähnte Textverarbeitung).

Wenn Sie den REM-Killer auf jedes in diesem Buch abgedruckte Programm anwenden, werden Sie manche Enttäuschung erleben, da ich in kurzen Programmen wie zum Beispiel "Zinzeszins", "Hardcopy" oder "Reaktionstest" bei der Programmierung nicht darauf achtete, ob Kommentarzeilen als Sprungziele verwendet werden.

Jene umfangreichen Programme, bei denen sich der Einsatz des REM-Killers zur Programmbeschleunigung anbietet (zum Beispiel die Textverarbeitung, der Etikettendruck, der Vokabeltrainer) lassen sich problemlos mit dem REM-Killer bearbeiten. In diesen Programmen werden keinerlei Kommentarzeilen als Sprungziele eingesetzt.

Programmbedienung

Der REM-Killer ist sehr einfach zu bedienen: Nach dem Starten werden Sie nach dem Namen des Programms gefragt, das bearbeitet werden soll. Wenn Sie den Namen eingegeben haben, müssen Sie noch angeben, ob das Programm von Cassette oder Diskette geladen werden soll. Geben Sie bitte "c" für Cassette beziehungsweise "d" für "Diskette" ein.

Der REM-Killer lädt das zu bearbeitende Programm nun in den Speicher des PLUS/4 und beginnt anschließend sofort mit seiner Arbeit. In der linken oberen Bildschirmecke sehen Sie die Nummern der zu löschenden Zeilen.

Wenn der Programmlauf beendet wurde, müssen Sie noch die "Überreste" des REM-Killers löschen, die sich gemeinsam mit dem bearbeiteten Programm im Speicher befinden. Da die letzte Zeilennummer des REM-Killers die 32 ist, geben Sie ein: "DEL -32". Anschließend befindet sich nur die verkürzte Programmversion im Speicher, die Sie nun abspeichern können.

```
1 rem ----(januar 1986/s.baloui)----
2 :
3 rem loeschen von zeilen, die mit
4 rem 'rem' beginnen oder (struktu-
5 rem rierung) nur ':' enthalten.
6 rem achtung: 'undef.statem.error'
7 rem bei sprungziel (goto, gosub)!
8 :
9 input"filename";n$
10 print "cassette/diskette (c/d) ?"
```

```
11 getkey a$
12 if a$="d" then a$="dload":z=10:else a$="load":z=12
13 scnclr
14 a=peek(45)+256*peek(46)-2:b=a-int(a/256)*256: c=
int(a/256)
15 print "poke 43,"b":poke 44,"c":new"
16 char 0,0,4,a$+chr$(34)+n$:rem char 0,0,4,"play" bei
cassette
17 char 0,0,z,"poke 43,"+str$(peek(43))+":poke 44,"
+str$(peek(44))
18 char 0,0,z+4,"run 21"
19 poke 1319,19:for i=1320 to 1326:poke i,13:next:poke
239,8:end
20 :
21 ma=peek(43)+256*peek(44)
22 :
23 p1=peek(ma+4):p2=peek(ma+5)
24 if p1=143 or (p1=58 and p2=0) then 27
25 ma=peek(ma)+256*peek(ma+1):if peek(ma) or peek(ma+1)
then 23:else end
26 :
27 a=int(ma/256):poke 217,a:poke 216,ma-256*a
28 zn=peek(ma+2)+256*peek(ma+3)
29 scnclr:print zn:print "run 32"
30 poke 1319,19:poke 1320,13:poke 1321,13:poke 239,3:end
31 :
32 ma=peek(216)+256*peek(217):goto 23
```

Programmablauf

Die folgende Erläuterung ist ohne Maschinensprachekenntnisse nur schwer zu verstehen. Ich muß im folgenden davon ausgehen, daß Ihnen bekannt ist, was unter "16-Bit-Adressierung" und "Zeigern" zu verstehen ist.

Ein BASIC-Programm wird vom BASIC-Interpreter folgendermaßen im Speicher des PLUS/4 abgelegt: Den Beginn jeder Programmzeile bilden zwei sogenannte "Link-Bytes", die einen Zeiger auf die nächste Programmzeile - genauer: auf deren

Link-Bytes - darstellen. Darauf folgen zwei Bytes, die die jeweilige Zeilennummer enthalten, und erst im Anschluß an diese insgesamt vier Bytes ist der eigentliche Programmtext der betreffenden Zeile gespeichert. Den Abschluß einer Programmzeile bildet das Byte null ($00), worauf die beiden Link-Bytes der nächsten Zeile folgen.

Link-Byte/Zeilennummer/Programmtext/$00/Linkbyte/...

Im eigentlichen Programmtext werden BASIC-Befehle in Form sogenannter "Tokens" abgelegt, ein Byte langer Schlüsselwörter. Das Token für das BASIC-Wort "REM" besteht aus dem Byte 143.

Das Prizip des REM-Killers beruht darauf, jede BASIC-Zeile der Reihe nach abzuarbeiten und sie zu löschen, wenn das erste Byte des Programmtextes entweder eine 143 (REM am Zeilenanfang = reine Kommentarzeile) ist, oder aber das erste Byte eine 58 (= Doppelpunkt) und das zweite Byte $00 ist (Doppelpunkt und danach Zeilenende = Zeile, die nur aus einem Doppelpunkt besteht).

Zum Löschen wird ein Trick verwendet: Mit Hilfe des Tastaturpuffers wird die Eingabe der jeweiligen Zeilennummer und das Drücken von "Return" im Direktmodus simuliert.

Dieser Trick wird bereits beim Nachladen des zu bearbeitenden Programms angewandt (siehe "Merge"): Die benötigten Befehle werden auf den Bildschirm geschrieben (Zeilen 15-18), in den Tastaturpuffer werden die Zeichen "Cursor home" und mehrmals "Return" "gepokt", in den Tastaturpufferzähler wird die Anzahl der "gepokten" Zeichen geschrieben, und der Programmbefehl "END" bewirkt nun die Abarbeitung des Tastaturpufferinhaltes (Zeile 19). Die im Tastaturpuffer enthaltenen Zeichen werden vom BASIC-Interpreter abgearbeitet, das zu bearbeitende Programm wird nachgeladen, und - da der letzte auf den Bildschirm geschriebene Befehl "RUN 21" lautet - der REM-Killer mit Zeile 21 gestartet.

In Zeile 21 wird zuerst der Zeiger auf den Anfang des BASIC-Programms (43/44) in der Variablen "ma" festgehalten. Danach wird der Inhalt des ersten und zweiten Bytes des Programmtextes der ersten Zeile ("ma+4" und "ma+5") in die Variablen "p1" und "p2" übertragen (Zeile 23). Wenn entweder das erste Byte dem Token für "REM" entspricht (143) oder aber das erste Byte einen Doppelpunkt darstellt (58) und das zweite Byte den Wert null besitzt (= Zeilenende), wird zu Zeile 27 gesprungen, der Zeilenlösch-Routine (Zeile 24).

Ansonsten wird die nächste Zeile bearbeitet, indem "ma" die Adresse der nächsten Link-Bytes erhält (Low-Byte + 256 * High-Byte). Enthalten die Link-Bytes der gerade bearbeiteten Zeile beide den Wert null, wurde das Programmende erreicht, und die Arbeit des REM-Killers ist beendet (Zeile 25).

In der Zeilenlösch-Routine, die ab Zeile 27 beginnt, wird zuerst der momentane Inhalt von "ma" in Low- und High-Byte aufgeteilt und in 216/217 gespeichert. Diese Speicherung ist nötig, da das Löschen im simulierten Direktmodus erfolgt und der REM-Killer anschließend - ebenfalls im simulierten Direkt-Modus - mit "RUN 32" wieder gestartet wird. Wie Sie alle wissen, gehen beim Starten eines Programms jedoch die Inhalte aller Variablen verloren.

Da der REM-Killer auch nach dem Löschen einer Zeile und dem anschließenden Neustart wissen muß, welche Adresse des BASIC-Programms zuletzt bearbeitet wurde, wird diese Adresse "ma" dauerhaft gespeichert, und beim Start des REM-Killers aus den Speicherstellen 216/217 wieder eingelesen (Zeile 32).

Das eigentliche Löschen erfolgt in den Zeilen 28-32. Die beiden den Link-Bytes folgenden Zeilen enthalten die jeweilige Zeilennummer. Diese wird auf den Bildschirm geschrieben und darunter der Befehl "Run 32". In den Tastaturpuffer werden die Zeichen "Cursor home" und zweimal "RETURN" geschrieben. In den Tastaturpufferzähler wird eine drei "gepokt", da sich im Tastaturpuffer drei Zeichen befinden, die nach dem Beenden des Programms abgearbeitet werden sollen (Zeile 30).

Der Tastaturpuffer wird nach dem Befehl "END" abgearbeitet, die Zeile gelöscht und der Befehl "RUN 32" ausgeführt, der den REM-Killer erneut startet.

Bitte verzeihen Sie mir, daß ich in diesem Programmlisting auf jegliche Kommentierung verzichtete und möglichst viele Befehle in eine Zeile packte. Oberste Priorität bei der Erstellung hatte jedoch die Programmlänge. Da der REM-Killer gemeinsam mit dem nachzuladenden Programm im Speicher vorhanden ist, sollte er selbst mit so wenig Platz und Programmzeilen wie möglich auskommen.

Folgende Schritte sind notwendig, um mit dem simulierten Direkt-Modus zu arbeiten:

1. Bildschirm löschen
2. Im Direkt-Modus zu bearbeitenden Befehl auf den Bildschirm schreiben
3. "Cursor home" in den Tastaturpuffer poken
4. "Return" in den Tastaturpuffer poken
5. Die Anzahl der zu bearbeitenden Zeichen (zwei) in den Tastaturpufferzähler poken
6. Mit "END" den Programm-Modus verlassen.

Soll ein Programm anschließend erneut gestartet werden, muß selbstverständlich auch der Befehl "RUN (Zeilennummer)" auf den Bildschirm geschrieben und ein weiteres "Return" in den Tastaturpuffer gepokt werden. Entsprechend muß der Wert für die Zeichenanzahl verändert werden, die in den Zähler geschrieben wird. Denken Sie bitte daran, daß alle Variablen durch den Befehl "RUN" gelöscht werden. Benötigte Variablen müssen in Low- und High-Bytes zerlegt direkt in bestimmten Speicherstellen gespeichert werden.

4.5.3. Merge

Unter "mergen" wird im EDV-Bereich die Verbindung zweier Programme verstanden. Merge-Routinen haben die Aufgabe, zu einem Programm, das sich bereits im Rechnerspeicher befindet, ein zweites dazuzuladen, ohne daß dadurch das erste Programm zerstört wird, wie es beim "LOAD"-Befehl der Fall wäre.

Mit Hilfe der vorgestellten Merge-Routine können Sie zum Beispiel Unterprogrammbibliotheken aufbauen. Sie können verschiedene Routinen (Sortierroutinen, Eingaberoutinen etc.) in ein bereits bestehendes Programm integrieren, ohne die jeweilige Routine Zeile für Zeile abtippen zu müssen.

Programmbedienung

Die Verwendung der Merge-Routine ist an zwei Bedingungen geknüpft: Zum einen muß selbstverständlich ausreichend Platz vorhanden sein für das Programm, das sich bereits im Speicher befindet, und für das zweite Programm, das "gemergt" werden soll.

Die zweite Bedingung besteht darin, daß sich die Zeilennummern beider Programme keinesfalls überschneiden dürfen. Sollte dieses der Fall sein, müssen Sie eines der beiden Programme vor dem "mergen" mit dem Befehl "RENUMBER" umnumerieren.

Gehen wir davon aus, daß beide Bedingungen erfüllt sind, und daß Sie das Programm mit den kleineren Zeilennummern bereits geladen haben. Geben Sie im Direktmodus ein:

```
print peek(43),peek(44)
```

Auf dem Bildschirm erscheinen zwei Zahlen, nennen wir Sie "a" und "b", die später noch benötigt werden. Üblicherweise hat "a" den Wert eins und "b" den Wert 16. Geben Sie als nächstes ein:

```
print peek(45)+256*peek(46)-2
```

Der ausgegebene Wert - nennen wir ihn vorläufig "x" - hängt von der Länge des Programms ab, das sich momentan im Speicher befindet. Die dritte Befehlszeile, die Sie eingeben müssen:

```
poke 43,x-int(x/256)*256:poke 44,int(x/256):new
```

Sollte "x" zum Beispiel den Wert 3590 besitzen, lautet die Befehlszeile:

```
poke 43,3590-int(3590/256)*256:poke 44,int(3590/256):new
```

Laden Sie nun das einzubindende Programm von Cassette oder Diskette und geben Sie anschließend ein:

```
poke 43,a:poke 44,b
```

Wenn "a" und "b" die üblichen Werte eins und sechzehn besitzen, lautet dieser Befehl:

```
poke 43,1:poke 44,16
```

Sie können das Programm nun listen. Unmittelbar hinter jenem Programm, das sich zuerst im Speicher befand, werden Sie das nachgeladene ("gemergte") zweite Programm entdecken. Die Befehlsfolge noch einmal zusammengefaßt:

Programmlisting

```
1. print peek(43),peek(44)              (ergibt "a" und "b")
2. print peek(45)+256*peek(46)-2               (ergibt "x")
3. poke 43,x-int(x/256)*256:poke 44,int(x/256):new
4. load"(programmname)  bzw.: dload"(programmname)
5. poke 43,a:poke 44,b
```

Beachten Sie, daß "a", "b" und "x" für die jeweiligen Zahlen stehen, die durch die "PRINT"-Befehle auf dem Bildschirm ausgegeben werden.

Programmablauf

Der BASIC-Interpreter des PLUS/4 besitzt zwei Zeiger, die auf den Anfang des BASIC-Programms (in 43/44) beziehungsweise auf den Anfang der Variablen (in 45/46) zeigen, die unmittelbar nach dem BASIC-Programm folgen und daher gleichzeitig dessen Ende (jedoch plus drei Byte) kennzeichnen. Das Prinzip der vorgestellten Routine besteht darin, den Zeiger auf den Anfang des BASIC-Textes zu verstellen, und zwar auf das Ende des BASIC-Programms.

Beim Laden wird ein Programm immer an jene Adresse geladen, auf die der Zeiger für den Anfang des BASIC-Textes weist. Da dieser jedoch so verstellt wird, daß er auf das Ende des Programms zeigt, wird das zu ladende Programm ab dieser Adresse, also hinter dem vorhandenen Programm geladen.

Nachdem das zweite Programm nachgeladen wurde, erhält der Zeiger wieder seinen Originalwert.

Der erste im Direktmodus eingegebene Befehl ermittelt die Adresse (Low/High), auf die der Zeiger momentan weist. Diese Adresse wird mit dem letzten Befehl wiederhergestellt, der Zeiger bekommt seinen ursprünglichen Wert.

Der zweite Befehl ermittelt die Adresse (minus zwei), auf die der Zeiger auf den Anfang der Variablen weist. Diese Adresse entspricht exakt der Endadresse des momentan vorhandenen Programms plus eins, weist also auf das erste Byte hinter dem Programm. Diese Adresse wird in Low- und High-Byte aufgeteilt und ergibt die neue Anfangsadresse des Zeigers auf den BASIC-Anfang. Nach dieser Korrektur ist unbedingt die Eingabe von "NEW" nötig. Erst nach diesem Befehl übernimmt der BASIC-Interpreter die neuen Zeigerwerte.

Das nachzuladende Programm wird nun dank des verstellten Zeigers unmittelbar hinter das bereits vorhandene Programm geladen.

4.5.4. Eingabe-Routine

Bei dem vorgestellten Programm handelt es sich um eine sogenannte "Eingabe-Routine". Eingabe-Routinen sind jene Programme, die jeder Programmierer üblicherweise als erstes schreibt, wenn er sich an einen neuen Computer wagt. Professionelles Programmieren ist ohne vernünftige Eingabe-Routinen nicht möglich. Das hier vorgestellte Programm dient als Ersatz für den im PLUS/4-BASIC vorhandenen "INPUT"-Befehl, der mehrere Unzulänglichkeiten aufweist:

1. Maximale Eingabelänge von 80 Zeichen
2. Keine Definition einer Eingabezone möglich
3. Keine Beschränkung der Eingabezeichen möglich

Wenn Sie selbst bereits mit dem "INPUT"-Befehl gearbeitet haben, werden Ihnen vor allem die beiden letzten Punkte bereits unangenehm aufgefallen sein. Sie können während der Eingabe jederzeit beliebig mit dem Cursor auf dem Bildschirm umherwandern, was im ungünstigsten Fall bei Beenden der Eingabe mit "RETURN" zu einer Fehlermeldung ("redo from start") führen kann.

Daß keine Beschränkung der Eingabezeichen möglich ist, ist sehr unangenehm, wenn als Eingabe beispielsweise nur Zahlen, jedoch keine Buchstaben zugelassen werden sollen. Unangenehm daran, daß jede beliebige Taste als gültige Eingabe zugelassen wird, ist auch, daß mit der Taste "Clr" der Bildschirm gelöscht und damit der mühsam aufgebaute Bildschirminhalt zerstört wird.

Mit der vorliegenden Routine wird der letzte und wichtigste von mir bemängelte Nachteil, die fehlende Möglichkeit, Eingaben auf bestimmte Tasten zu beschränken, aufgehoben. Sie können einen String definieren, der aus beliebigen Zeichen bestehen darf. Alle in diesem String vorhandenen Zeichen werden bei Ablauf der Eingabe nicht angenommen, die Betätigung der entsprechenden Tasten hat keinerlei Auswirkungen.

Programmbedienung

Bauen Sie die Input-Routine am besten am Beginn eines Programms an, das heißt laden Sie die Routine vor der eigentlichen Programmerstellung und schreiben Sie Ihr Hauptprogramm ab Zeile 470. Wenn Sie wollen, können Sie die Routine selbstverständlich auch hinter Ihr eigentliches Hauptprogramm schieben und die Routine umnumerieren, zum Beispiel mit "RENUMBER 10000,10".

Die eigentliche Eingaberoutine beginnt ab Zeile 350. Zuvor müssen jedoch die benötigten Variablen "cs", "cz", "cr", "sc" und "nz$" definiert worden sein (am besten am Programmanfang).

Im folgenden Programmlisting enthält "nz$" die Steuerzeichen "Cursor home" (= chr$(19)), "Clear" (= chr$(147)), das Anführungszeichen (= chr$(48)), ein Komma und einen Doppelpunkt.

Alle diese Zeichen sind als Eingabetasten nicht zulässig. Während einer Eingabe kann daher weder der Bildschirm gelöscht (= "Clear"), noch der Cursor auf die linke obere Bild-

schirmecke gesetzt werden (= "Cursor home"), noch ist es möglich, eines der übrigen in "nz$" enthaltenen Zeichen einzugeben.

Diese Definition von "nz$" soll selbstverständlich nur als Beispiel dienen. Ich empfehle Ihnen jedoch auf alle Fälle, die Tasten "Cursor home" und "Clear" nie zuzulassen, da deren Betätigung während einer Eingabe kaum sinnvoll sein kann.

Aufgerufen wird die Routine mit "GOSUB 350". Nach dem Aufruf erscheint der Cursor (der invers, jedoch nicht blinkend dargestellt wird), und Sie können jedes - außer den in "nz$" enthaltenen - Zeichen eingeben. "RETURN" beendet eine Eingabe, die dem aufrufenden Programm in "e$" übergeben wird. Wie Sie sehen, arbeitet die Routine analog dem "INPUT"-Befehl, so daß keinerlei Umstellungen für den Benutzer notwendig sind.

Programmlisting

```
100 rem -----eingabe-routine-----
110 rem -(januar 1986/s.baloui)--
120 :
130 rem ---variablen---
140 rem cs=cursorspalte
150 rem cz=cursorzeile
160 rem sc=startadresse bildschirm
170 rem nz$=nicht zugelassene zeichen
180 rem e$=eingabe
190 rem s/z=spalte/zeile vom eingabestart
200 rem a=momentane cursorposition
210 rem b=zeichen unter cursor
220 :
230 cs=202
240 cz=205
250 cr=55464
260 sc=3072
270 :
280 nz$=chr$(19)+chr$(147)+chr$(48)+",:"
290 gosub 350:rem aufruf der routine
300 print e$:rem eingabe
```

```
310 end
320 :
330 :
340 rem ---eingabe-unterroutine---
350 s=peek(cs):z=peek(cz)
360 :
370 a=sc+peek(cs)+40*peek(cz):b=peek(a)
380 poke a,b+128
390 getkey e$:if instr(nz$,e$) then 390
400 poke a,b
410 poke 207,0:rem 0 inserts
420 if e$<>chr$(13) then print e$;:goto 370
430 :
440 poke cs,s:poke cz,z:sys cr
450 open 3,3:input#3,e$:close 3
460 return
```

Programmablauf

Nach dem Aufruf der Routine werden zuerst die momentane Spalten- und Zeilenposition des Cursors in den Variablen "s" und "z" gespeichert (Zeile 350).

In Zeile 370 wird die Speicherstelle im Bildschirmspeicher errechnet, die der momentanen Cursorposition entspricht. Sie ergibt sich aus der Anfangsadresse des Bildschirmspeichers ("sc") plus der Cursorspalte ("peek(cs)") plus 40 mal der Cursorzeile ("peek(cz)"). Der Inhalt "b" dieser Speicherstelle ist der Code, der dem Zeichen unter dem Cursor zugeordnet ist.

In die ermittelte Speicherstelle wird der um 128 erhöhte Code "gepokt", der dem gleichen Zeichen in reverser Darstellung entspricht (Zeile 380). Durch die Invers-Darstellung des Zeichens an der momentanen Cursorposition ergibt sich ein - wenn auch nicht blinkender - inverser Cursor, der dem Benutzer die momentane Eingabeposition mitteilt.

In Zeile 390 wird nun auf die Betätigung einer Taste gewartet, diese jedoch ignoriert, wenn das zugehörige Zeichen in "nz$" enthalten ist.

Wenn Zeile 390 verlassen wurde, das heißt wenn ein zulässiges Zeichen eingegeben wurde, wird in die Speicherstelle "a" der ursprüngliche Wert des Zeichens unter dem Cursor "gepokt", dieses wird dadurch wieder normal dargestellt.

Diese "Normalisierung" ist nötig, da sich der Cursor nach der Eingabe eines Zeichens weiterbewegt und es unbefriedigend wäre, wenn an der alten Cursorposition ein invers dargestelltes Zeichen zurückbliebe.

In die Speicherstelle 207 wird nun der Wert null "gepokt", bevor das eingegebene Zeichen auf dem Bildschirm ausgegeben wird (außer, die Taste "RETURN" wurde gedrückt und die Eingabe damit beendet).

Diese Speicherstelle enthält die sogenannte "Anzahl ausstehender Inserts". Wie Sie wissen, erscheinen auf dem Bildschirm unverständliche Steuerzeichen, wenn Sie "INST" drücken und anschließend den Cursor bewegen. In der Speicherstelle 207 ist die Anzahl der noch nicht bearbeiteten Inserts enthalten, die darüber entscheidet, wie viele folgende Cursorbewegungen statt zur Bewegung des Cursors zur Ausgabe von Steuerzeichen führen.

Wenn diese Speicherstelle immer vor der Ausgabe eines Zeichens auf null gesetzt wird, sind Sie das leidige Problem der Steuerzeichen los. Jedes ausgegebene Zeichen wird normal behandelt, auch dann, wenn unmittelbar zuvor "INST" betätigt wurde.

Wenn die Taste "RETURN" betätigt wird, erfolgt in Zeile 420 kein Sprung zum Beginn der Eingabeschleife, sondern die Eingabe wurde beendet und wird nun vom Bildschirm in "e$" eingelesen.

In Zeile 440 wird der Cursor auf die ursprüngliche Position vor Beginn der Eingabe gesetzt. Die in "s" und "z" gerettete Spalten- und Zeilenposition wird in die entsprechenden Speicherstellen

"gepokt", und anschließend wird der Cursor durch Aufruf einer im Betriebssystem des PLUS/4 eingebauten Routine ("sys cs" = "sys 55464") auf diese Ausgangsposition gesetzt.

In Zeile 450 wird der Bildschirm als Eingabegerät definiert (Gerätenummer drei). Der anschließende "INPUT"-Befehl bezieht sich dann nicht wie üblich auf die Tastatur, sondern auf das neue Eingabegerät, den Bildschirm. Der "INPUT"-Befehl liest die Eingabe des Benutzers in den String "e$" ein. Zuletzt wird der geöffnete Kanal wieder geschlossen und die Eingabe-Routine verlassen.

4.5.5. Shape-Editor

Wie Ihnen bekannt ist, kann der PLUS/4 Graphikfiguren verwalten, die sogenannten "Shapes". Shapes können mit einfachen Bewegungen auf dem Graphikbildschirm bewegt werden und eignen sich daher hervorragend zur Spiele-Programmierung.

Die Konstruktion eines Shapes stellt jedoch ein größeres Problem dar. Die einzige Möglichkeit besteht in langwierigen Versuchen mit den vorhandenen Graphikbefehlen. Sie müssen ein Programm schreiben, das eine Figur zeichnet, zum Ändern der Figur das Programm selbst ändern und neu starten. Dieser Vorgang muß wiederholt werden, bis sich die gewünschte Figur ergibt, die anschließend mit dem Befehl "SSHAPE" in eine Stringvariable übertragen werden kann.

Das Programm "Shape-Editor" ermöglicht die interaktive Shape-Konstruktion. Sie malen das Shape auf den Bildschirm, verändern Ihre Zeichnung beliebig und geben den Befehl zum Abspeichern des Shapes, wenn Sie mit der Zeichnung einverstanden sind.

Programmbedienung

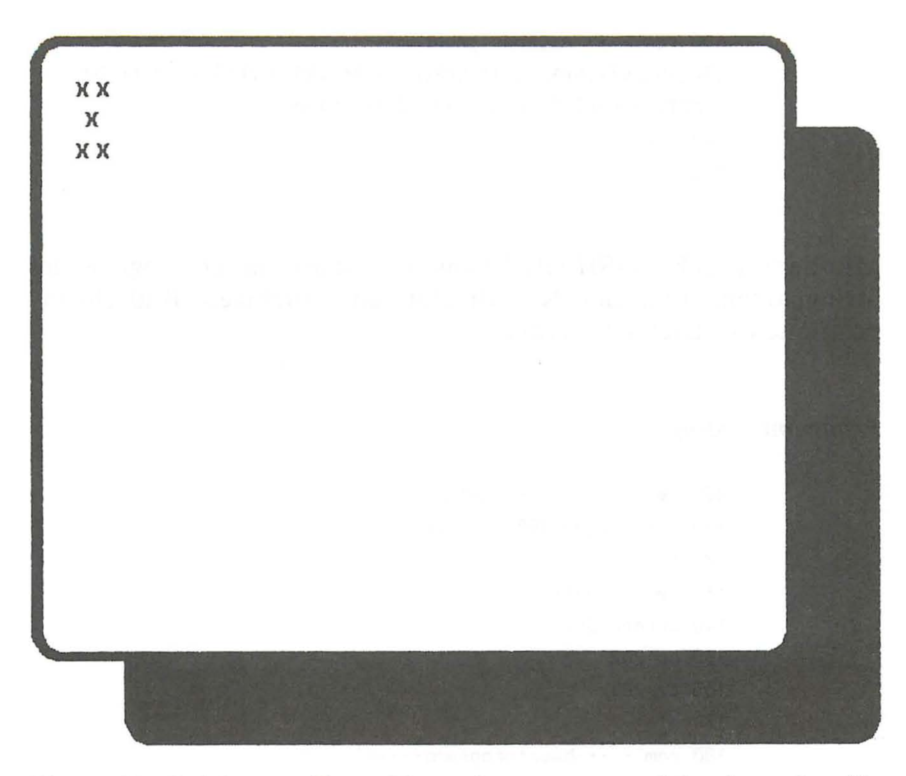
Zum Editieren des Shapes wird der normale Textbildschirm verwendet. Jedes Zeichen entspricht einem Punkt des Shapes. Die maximale Ausdehnung der editierten Shapes beträgt daher 40 Punkte in der X- und 25 Punkte in der Y-Richtung.

Nach dem Starten des Programms sehen Sie den invertiert dargestellten Cursor. Sie können ihn mit den vier Cursortasten und der Taste "Home" beliebig über den Bildschirm bewegen. Mit der Taste "Clear" kann eine unbefriedigende Zeichnung komplett gelöscht werden.

Die späteren Shape-Punkte können Sie mit einem beliebigen Zeichen setzen, zum Beispiel mit "x". Mit der Leertaste können falsch gesetzte Punkte wieder gelöscht werden.

Das in diesem Buch vorgestellte Mal-Programm verwendet ein Shape, das die Form eines "x" besitzt und dessen Ausdehnung sowohl in der X- als auch der Y-Richtung jeweils drei Punkte beträgt.

Um dieses Shape zu erstellen, müssen auf dem Bildschirm folgende Punkte markiert werden:



Wenn die Zeichnung Ihren Vorstellungen entspricht, beenden Sie die Editierung des Shapes, indem Sie "RETURN" drücken. Die hochauflösende Graphik wird eingeschaltet und das Shape gezeichnet. Mit dem Befehl "SSHAPE" wird die jeweilige Punktekonfiguration in die Variable "shape$" übertragen. Die ASCII-Werte der einzelnen Zeichen, aus denen diese Variable besteht, werden auf dem Bildschirm ausgegeben.

Schreiben Sie diese Werte bitte auf. Mit ihrer Hilfe können Sie das erstellte Shape später jederzeit in Ihre eigenen Programme einbauen. Das im Beispiel erstellte Shape besteht aus folgenden ASCII-Werten, die das Programm ausgibt:

```
160  64  160  2  0  2  0
```

Um das Shape in Programme einzubauen, sollten Sie am Programmanfang einen String definieren, der aus den zu diesen ASCII-Codes gehörenden Zeichen besteht:

```
100 rem -----eigenes programm-----
110 shape$=chr$(160)+chr$(64)+chr$(160)+chr$(2)+chr$(0)
+chr$(2)+chr$(0):rem shape-definition
120 ...
130 ...
140 ...
```

Mit dem Befehl "GSHAPE" kann das Shape unter Angabe des Stringnamens und der Koordinaten an beliebigen Bildschirmpositionen gezeichnet werden.

Programmlisting

```
100 rem -----shape-editor-----
110 rem --(jan 1986/s.baloui)-
120 :
130 rem ---variablen---
140 screen=3072
150 cs=202
160 cz=205
170 :
180 rem -----hauptprogramm-----
190 scnclr
200 a=peek(cs)+40*peek(cz)+screen
210 poke a,peek(a) or 128:rem zeichen invert.
220 getkey a$
230 if a$=chr$(13) then poke a,peek(a) and 127:goto 290
240 print a$;
250 poke a,peek(a) and 127:rem zeichen normalisieren
260 goto 200
270 :
280 rem ---shape zeichnen---
290 graphic 1,1
300 :
310 z=screen
320 for i=0 to 24
330 for ii=0 to 39
```

```
340 if peek(z)=32 then 380
350 draw 1,160+ii,100+i
360 if i>zm then zm=i
370 if ii>sm then sm=ii
380 z=z+1
390 next ii,i
400 :
410 rem ---sshape/daten ausgeben---
420 sshape shape$,160,100,160+sm,100+zm
430 graphic 0
440 scnclr
450 print "shape-daten:"
460 :
470 for i=1 to len(shape$)
480 print asc(mid$(shape$,i,1));
490 next
```

Programmablauf

Wie Sie sehen, ist das Programm "Shape-Editor" außerordentlich kurz. Es besteht aus den zwei Teilen "Editieren des Shapes" (Zeile 180-260) und "Übernahme des Shapes" (Zeile 290-490).

Das Editieren geschieht in einer Schleife. Zu Beginn der Schleife wird das Zeichen, das sich an der momentanen Cursorposition befindet, invers dargestellt. Die zugehörige Position im Bildschirmspeicher wird über dessen Startadresse "screen", die Cursorspalte "cs" und die Cursorzeile "cz" errechnet (Zeile 200). Die Zahl in der ermittelten Speicherstelle wird mit "OR 128" verknüpft (Zeile 210). Diese Verknüpfung ergibt den Code des invers dargestellten Zeichens.

Nun wird auf die Betätigung einer Taste gewartet. Das Zeichen wird ausgegeben (Zeile 220) und das Zeichen an der bisherigen Cursorposition wieder normalisiert (Zeile 250), das heißt die Inversdarstellung wird rückgängig gemacht.

Diese Schleife wird erst verlassen, wenn der Benutzer "Return" drückt, das Editieren des Shapes beendet ist.

Nun wird die hochauflösende Graphik eingeschaltet und der Textbildschirmspeicher Zeile um Zeile abgetastet, wobei jeweils 40 Spalten überprüft werden (Zeile 320-330). Immer dann, wenn sich an der momentan abgetasteten Speicherstelle ein anderer als der Code 32 (= Leertaste) befindet (Zeile 340), wird an der entsprechenden Stelle des Graphikbildschirms ein Punkt gesetzt (Zeile 350).

Bei diesem Abtasten wird zugleich die maximale Ausdehnung des Shapes in X- beziehungsweise in Y-Richtung ermittelt, das heißt die höchste Spalten- beziehungsweise Zeilennummer, in der ein Punkt gesetzt wurde (Zeile 360-370).

Der geschilderte Vorgang wird beendet, wenn der gesamte Textbildschirm (-speicher) Zeichen für Zeichen abgetastet wurde.

Das Shape wird nun in die Variable "shape$" übertragen (Zeile 420). Die Graphik wird ausgeschaltet und dem Benutzer werden die ASCII-Codes der einzelnen Zeichen ausgegeben, aus denen "shape$" und damit das erstellte Shape besteht.

4.6. Sonstige Programme

Im folgenden werden verschiedene Programme vorgestellt, die sich nicht eindeutig einer der behandelten Kategorien zuordnen lassen.

4.6.1. Etikettendruck

Mit dem vorliegenden Programm können beliebig viele überall erhältliche Standardetiketten mit einem von Ihnen festzulegenden Aufdruck "am Fließband" bedruckt werden. Dieses etwas umfangreichere Programm kann problemlos mit dem REM-Killer gekürzt werden.

Programmbedienung

Nach dem Programmstart erhalten Sie auf dem Bildschirm die wichtigsten Informationen zur Programmbedienung. Der Cursor befindet sich in der ersten von acht Eingabezeilen. Jede dieser Zeilen entspricht einer Etikettenzeile. Tragen Sie in die einzelnen Zeilen den gewünschten Aufdruck ein. Mit den Tasten "Cursor rechts", "Cursor links", "Del" und "Inst" können Eingaben innerhalb eines Feldes editiert werden. Mit den Tasten "Cursor nach unten" und "Cursor nach oben" können Sie sich beliebig von Feld zu Feld bewegen.

Eingabebeispiel:

1 Data Becker GmbH
2 Vetriebsabteilung
3 Leiter: Herr XYZ
4 Tel.: 0211/3100-123
5
6 Merowingerstr. 30
7
8 4000 Duesseldorf

Beenden Sie die Eingabe, indem Sie eine beliebige Funktionstaste drücken. Anschließend werden Sie nach der Anzahl der auszudruckenden Etiketten gefragt. Geben Sie die gewünschte Zahl ein. Sie können den Druckvorgang nun starten, indem Sie eine Taste betätigen. Vergewissern Sie sich zuvor jedoch unbedingt, ob die Etiketten richtig justiert sind! Richtig justiert sind die Etiketten, wenn sich der Druckkopf unmittelbar unter der Oberkante einer Etikette befindet.

Programmlisting

```
100 rem ----etikettendruck-----
110 rem -(januar 1986/s.baloui)-
120 :
130 rem --variablen--
140 rem cs=cursorspalte
150 rem cz=cursorzeile
160 rem cr=cursorzeile
170 rem fa=max.feldanzahl
180 rem ft$=zeichen, mit denen die funktionstasten neu
belegt wurden
190 rem cr$ bis ra$=ascii-codes der steuertasten in
stringform
200 rem ed$=tasten, mit denen eine eingabe editiert werden
kann
210 rem ez$=tasten, mit denen die eingabe in ein feld
beendet wird
220 rem z$=zeichen, die als gueltige eingabe zugelassen
werden
230 rem fb$(...)=feldbezeichnungen
240 rem u$(...)=feldinhalte
250 rem lm(...)=feldlaengen
260 rem po=position im eingabefeld
270 rem lm=laenge des aktuellen feldes
280 rem li=nummer des aktuellen feldes
290 rem fl=zeile des 1.eingabefeldes
300 rem ae=anzahl zu druckender etiketten
310 :
320 rem --dimensionierung/cursor--
330 fa=20:rem maximale feldanzahl
340 dim u$(fa),fb(fa),lm(fa)
350 cs=202
360 cz=205
370 cr=55464
380 :
390 rem --funktionstastenbelegung--
400 for i=1 to 8
410 f$(i)=chr$(i+132)
420 ft$=ft$+f$(i)
```

```
430 key i,f$(i)
440 next
450 :
460 rem --steuerzeichen--
470 cr$=chr$(29):rem cursor rechts
480 cl$=chr$(157):rem cursor links
490 cu$=chr$(17):rem cursor unten
500 co$=chr$(145):rem cursor oben
510 ho$=chr$(19):rem taste 'home'
520 del$=chr$(20):rem taste 'del'
530 inst$=chr$(148):rem taste 'inst'
540 re$=chr$(18):rem revers ein
550 ra$=chr$(146):rem revers aus
560 :
570 rem --versch.strings--
580 ed$=cr$+cl$+del$+inst$
590 ez$=cu$+co$+ft$
600 z$="1234567890!#$%&'()qwertyuiopasdfghjklzxcvbnmQWERTY
UIOPASDFGHJKLZXCVBNM"
610 z$=z$+"^ +-*/.,;<>[]?="
620 a$="          ":for i=1 to 25:sp$=sp$+a$:next:rem 25*10
spaces
630 :
640 rem --maskenbeschreibung--
650 mf=8
660 for i=1 to mf
670 fb$(i)=left$(str$(i)+" ",2)+":  "
680 lm(i)=30
690 u$(i)=left$(sp$,lm(i))
700 next
710 :
720 goto 1250:rem hauptprogramm
730 :
740 :
750 rem -----eingabe-routine-----
760 po=1
770 print re$ mid$(u$(li),po,1) ra$ cl$;
780 getkey a$
790 if instr(ez$,a$) then printmid$(u$(li),po,1):return
800 on instr(ed$,a$) goto 870,910,950,1010
```

```
810 if instr(z$,a$)=0 or po>=lm then 770
820 :
830 rem --normales zeichen--
840 mid$(u$(li),po,1)=a$:po=po+1:print a$;:goto 770
850 :
860 rem --cursor links--
870 if po<lm then printmid$(u$(li),po,1);:po=po+1
880 goto 770
890 :
900 rem --cursor rechts--
910 if po>1 then printmid$(u$(li),po,1);:po=po-1:
printcl$cl$;
920 goto 770
930 :
940 rem --delete--
950 s=peek(cs):z=peek(cz)
960 h$=mid$(u$(li),po+1):printh$;:mid$(u$(li),po)=h$+" "
970 poke cs,s:poke cz,z:sys cr
980 goto 770
990 :
1000 rem --insert--
1010 if po=lm then 1050
1020 s=peek(cs):z=peek(cz)
1030 h$=" "+mid$(u$(li),po,lm-po): printh$;: u$(li)=
mid$(u$(li),1,po-1)+h$
1040 poke cs,s:poke cz,z:sys cr
1050 goto 770
1060 :
1070 :
1080 rem -----maskengenerator-----
1090 for i=1 to mf
1100 poke cs,0:poke cz,i+fl:sys cr
1110 print fb$(i) u$(i)
1120 next
1130 :
1140 li=1
1150 lm=lm(li)
1160 poke cs,len(fb$(li)):poke cz,li+fl:sys cr
1170 gosub 760
1180 if instr(ft$,a$) then return
```

```
1190 if a$=cu$ then if li<mf then li=li+1
1200 if a$=co$ then if li>1 then li=li-1
1210 goto 1150
1220 :
1230 :
1240 rem ---------hauptprogramm---------
1250 print chr$(14):rem gross/klein
1260 scnclr
1270 print tab(5)"Etikettendruck"
1280 print tab(5)"--------------"
1290 print:print:print"Tragen Sie bitte den Etiketten
aufdruck"
1300 print"in die einzelnen Felder ein und druecken"
1310 print"Sie danach eine beliebige Funktionstaste"
1320 fl=peek(cz)+1
1330 gosub 1090
1340 :
1350 poke cs,0:poke cz,18:sys cr
1360 print "Wieviele Etiketten sollen bedruckt"
1370 input"werden";ae
1380 print:print "Justieren Sie die Etiketten (Druckkopf"
1390 print "an Etikettenoberkante) und druecken"
1400 print "Sie eine beliebige Taste"
1410 getkey a$
1420 :
1430 open 4,4,7
1440 for i=1 to ae
1450 for ii=1 to 12
1460 print#4,u$(ii)
1470 nextii,i
1480 close4
1490 run
```

Programmablauf

Kern des Programms ist die Eingaberoutine, die im Kapitel über die integrierte Text- und Dateiverwaltung ausführlich beschrieben wird. Die einzigen Unterschiede betreffen die fehlende - da unnötige - Verwaltung mehrerer Bildschirmseiten

und das Fehlen komfortabler Funktionen wie zum Beispiel "Zeile löschen" oder "Tabulatorsprung".

Nachdem der Benutzer Informationen zur Programmbedienung erhielt (Zeile 1240-1320), wird die Eingaberoutine aufgerufen. Die auszudruckenden Strings sind nach dem Beenden der Eingabe in "u$(1)" bis "u$(8)" enthalten.

Der Benutzer wird nun nach der Anzahl der auszudruckenden Etiketten gefragt und der eingegebene Wert in der Variablen "ae" gespeichert (Zeile 1360-1370).

Der eigentliche Ausdruck erfolgt in den Zeilen 1430-1480. Zuerst wird ein Übertragungskanal zum Drucker geöffnet (Zeile 1430). Wichtig ist hierbei die Verwendung der Sekundäradresse sieben! Diese Sekundäradresse schaltet Commodore-Drucker in den "Klein/Großschreib-Modus". Fehlt die Angabe dieser Sekundäradresse, werden nur Großbuchstaben gedruckt (siehe Druckerhandbuch).

Der Ausdruck erfolgt in zwei Schleifen: Einer äußeren Schleife, die von eins bis "ae" läuft, bis zur letzten auszudruckenden Etikette, und einer inneren Schleife, die von eins bis zwölf läuft und für den Ausdruck einer einzelnen Etikette zuständig ist. Der Endwert dieser Schleife ist nur dann verständlich, wenn man weiß, daß der Abstand zwischen den Oberkanten zweier Standardetiketten zwölf Druckzeilen (bei normalem Zeilenabstand) beträgt.

In der inneren Schleife werden die Strings "u$(1)" bis "u$(12)" ausgedruckt, wobei nur die ersten acht Strings vom Benutzer eingegebene Informationen enthalten können. Der Ausdruck der restlichen vier leeren Strings dient zum korrekten Vorschub des Druckkopfes zur Oberkante der nächsten Etikette.

Nachdem die gewünschte Etikettenanzahl bedruckt wurde, wird der Übertragungskanal geschlossen und das Programm mit "RUN" gestartet (Zeile 1490). Der Benutzer kann - wenn er es wünscht - erneut Etiketten mit beliebigen Informationen bedrucken.

4.6.2. Lottozahlen

Dieses Programm ermittelt Zufallszahlen zwischen eins und 49. Außer einer einzelnen Ausspielung können Sie sich auch beliebig viele ausgeben lassen und anschließend eine "Ausspielung" auswählen.

Programmbedienung

Das Programm ermittelt sofort nach dem Start sechs Zahlen und die Zusatzahl. Wenn Sie es wünschen, wird anschließend eine weitere Ausspielung simuliert.

Programmlisting

```
100 rem -----lottozahlen-----
110 rem (januar 1986/s.baloui)
120 :
130 rem ---variablen---
140 rem rz=zufallszahl
150 rem rz(...)=bereits ermittelte zahlen
160 :
170 rem -----hauptprogramm-----
180 print chr$(14):rem gross/klein
190 for i=1 to 7
200 rz=int(rnd(0)*48)+1
210 for ii=1 to i
220 if rz=rz(ii) then 200
230 next ii
240 rz(i)=rz
250 if i=7 then print "Zusatzzahl:"rz(i):goto 270
260 print rz(i),
270 next i
280 :
```

```
290 print:print " Noch eine Ausspielung (j/n) ?"
300 getkey a$
310 if a$="j" then run
```

Programmablauf

Die Zufallszahlenermittlung geschieht in einer Schleife, die vom Startwert eins bis zum Endwert sieben läuft (Zeile 190-270). Die ermittelten Zahlen werden in dem Array "rz(...)" gespeichert.

Jede neue Zahl wird mit allen bereits gespeicherten Zahlen verglichen (Zeile 210-230).Ist die ermittelte Zahl in dem Array bereits vorhanden, wird erneut eine Zufallszahl ermittelt, da jede Zahl nur einmal vorkommen darf.

Die Zufallszahlen werden auf dem Bildschirm ausgegeben, und - falls "i" den Wert sieben besitzt - dem Benutzer mitgeteilt, daß es sich um die Zusatzzahl handelt (Zeile 250).

4.6.3. Datumsberechnung

"Datumsberechnung" wurde freundlicherweise von V.Bühn zur Verfügung gestellt. Der Programmablauf wird nicht erläutert.

Das vorliegende Programm besitzt zwei Funktionen. Zum einen können Sie ein beliebiges Datum eingeben und das Programm ermittelt Ihnen den zugehörigen Wochentag. Zweitens ist das Programm in der Lage, die Differenz zweier Daten zu berechnen.

Programmbedienung

Nach dem Programmstart erscheint das Auswahlmenü. Drücken Sie die Taste "1" zur Ermittlung eines Wochentages und die Taste "2" zur Berechnung der Differenz zweier Daten. Geben Sie ein Datum bitte immer in der Reihenfolge "Tag", "Monat", "Jahr" ein.

Beispiel:

```
Tag? 10
Monat? 2
Jahr? 1986
```

Programmlisting

```
100 rem -----datumsberechnung-----
110 rem ------(1985/v.buehn)------
120 :
130 printchr$(14):rem gross/klein
140 co$=chr$(145)
150 no$="                                        "
160 t1$="Samstag   Sonntag   Montag    Dienstag  Mittwoch
DonnerstagFreitag"
170 a$=" 0 1-1 0 0 1 1 2 3 3 4 4"
180 b$="312931303130313130313031"
190 goto 550
200 :
210 rem ---datumseingabe---
220 scnclr:input"Tag";t:if t>31 or t<1 then 220
230 input"Monat";m:if m>12 or m<1 then gosub 500:goto 230
240 input"Jahr";j:if j<1500 or j>3000 then gosub 500:goto
240
250 b=val(mid$(b$,2*m-1,2))
260 if t>b then scnclr:gosub 500:goto 220
270 if j/4<>int(j/4) and m=2 and t>28 then scnclr:gosub
480:goto 220
280 if (j=1700orj=1800orj=1900) and t=29and m=2 then
scnclr:gosub 480:goto 220
290 a=val(mid$(a$,2*m-1,2))
300 if j/4=int(j/4) and m>2 then a=a+1
310 if j=1700 or j=1800 or j=1900 and m>2 then a=a-1
320 x=t+30*(m-1)+a
330 print:print "Das ist der"x". Tag im Jahr und ein"
340 j1=j-1584:xj=int((j1-1)/4)+1:if j=1584 then xj=0
```

```
350 x3=365*j1+xj+x
360 if j>1700 then x3=x3-1:rem korrekturen
370 if j>1800 then x3=x3-1:rem ""
380 if j>1900 then x3=x3-1:rem ""
390 x1=int((x3/7-int(x3/7))*7+.5):rem x1 ist der rest
modulo7
400 print mid$(t1$,10*x1+1,10)
410 print:print:print"Weitermachen (j/n) ?"
420 getkey g$
430 if g$="j" then 190
440 if g$="n" then end
450 goto 420
460 :
470 rem ---fehlermeldungen ausgeben---
480 print:print:print"Dieses Jahr war kein Schaltjahr!"
490 for i=1 to 500:next
500 print no$"Falsche Eingabe!"
510 for i=1 to 1000:next
520 print co$ no$ co$:return
530 :
540 rem -----hauptprogramm-----
550 scnclr:print:print tab(4)"Programm zur Berechnung von "
560 print tab(4)"Wochentagen und Zeitdifferenzen"
570 print:print "Bitte immer zuerst den Tag (1-31),"
580 print "danach den Monat (1-12) und zuletzt das"
590 print "Jahr (1500-3000) eingeben"
600 print:print:print
610 print tab(3)"(1)  Wochentag berechnen"
620 print tab(3)"(2)  Differenz zweier Zeitpunkte"
630 getkey g$
640 if g$="1" then 220
650 if g$="2" then 690
660 goto 630
670 :
680 rem ---differenzen berechnen---
690 scnclr:print "Die Differenz zweier Daten wird hier
berechnet"
700 print:print "Erstes Datum eingeben!"
710 input"Tag";t1:if t1>31 or t1<1 then gosub 500:goto 710
```

```
720 input"Monat";m1:if m1>12 or m1<1 then gosub 500:goto
720
730 input"Jahr";j1:if j1<1500 or j1>3000 then gosub 500:
goto 730
740 b=val(mid$(b$,2*m1-1,2))
750 if t1>b then scnclr:gosub 500:goto 710
760 if j1/4<>int(j1/4) and m1=2 and t1>28 then scnclr:gosub
480:goto 710
770 if(j1=1700orj1=1800orj1=1900)andt1=29andm1=2 then
scnclr:gosub 480:goto 710
780 print:print "Zweites Datum eingeben!"
790 input"Tag";t2:if t2>31 then 790
800 input"Monat";m2:if m2>12 or m2<1 then gosub 500:goto
800
810 input"Jahr";j2:if j2<1500 or j2>3000 then gosub 500:
goto 810
820 b=val(mid$(b$,2*m2-1,2))
830 if t2>b then scnclr:gosub 500:goto 790
840 if j2/4<>int(j2/4) and m2=2 and t2>28 then scnclr:gosub
480:goto 790
850 if(j2=1700orj2=1800orj2=1900)andt2=29andm2=2 then
scnclr:gosub480:goto790
860 a1=val(mid$(a$,2*m1-1,2))
870 if j1/4=int(j1/4) and m1>2 then a1=a1+1
880 if j1=1700 or j1=1800 or j1=1900 then a1=a1-1
890 x1=t1+30*(m1-1)+a1
900 ja=j1-1584:xa=int((ja-1)/4)+1:if j1=1584 then xa=0
910 xb=365*ja+xa+x1
920 if j1>1700 then xb=xb-1:rem korrektur
930 if j1>1800 then xb=xb-1:rem ""
940 if j1>1900 then xb=xb-1:rem ""
950 a2=val(mid$(a$,2*m2-1,2))
960 if j2/4=int(j2/4) and m2>2 then a2=a2+1
970 if j2=1700 or j2=1800 or j2=1900 then a2=a2-1
980 x2=t2+30*(m2-1)+a2
990 ka=j2-1584:ya=int((ka-1)/4)+1:if j2=1584 then ya=0
1000 yb=365*ka+ya+x2
1010 if j2>1700 then yb=yb-1:rem korrektur
1020 if j2>1800 then yb=yb-1:rem ""
1030 if j2>1900 then yb=yb-1:rem ""
```

```
1040 print:print "Die Differenz der beiden Daten ist":print
abs(yb-xb)"Tage"
1050 print:print "Weitermachen (j/n) ?"
1060 getkey g$
1070 if g$="j" then 190
1080 if g$="n" then end
1090 goto 1060
```

5. Die wichtigsten verwendeten Tricks

Für jene Leser, denen das Studium der Programmabläufe zu aufwendig ist, fasse ich die wichtigsten der verwendeten Tricks in diesem Kapitel zusammen. Bei Verständnisschwierigkeiten muß ich Sie jedoch bitten, die ausführlichen Erläuterungen zu den verschiedenen Programmen nachzulesen. In diesen erfahren Sie auch weitere Hinweise zu speziellen Besonderheiten, die in diesem Kapitel nicht erwähnt werden. An dieser Stelle werden nur Tricks aufgeführt, die möglichst kurz und mit wenigen Programmzeilen ausführbar sind.

5.1. Cursor setzen

Mit dem "CHAR"-Befehl ist es zwar möglich, an einer beliebigen Bildschirmposition Zeichen auszugeben. Es fehlt jedoch ein Befehl, der es erlaubt, ohne folgende Ausgaben den Cursor direkt auf eine beliebige Bildschirmposition zu setzen (zum Beispiel, um anschließend aus der zugehörigen Position im Bildschirmspeicher mit "PEEK" das Zeichen an der momentanen Bildschirmposition zu lesen).

In der sogenannten "Zeropage" des PLUS/4 existieren zwei Speicherstellen, in denen die Spalte und Zeile der aktuellen Cursorposition festgehalten ist (Spalte in 202, Zeile in 205). Durch Veränderung der Inhalte dieser Speicherstellen kann der Cursor auf eine beliebige Position gesetzt werden. Anschließend muß jedoch noch eine Betriebssystemroutine mit "SYS 55464" aufgerufen werden. Erst nach Aufruf dieser Routine wird der Cursor auf die in den beiden Speicherstellen enthaltene Position gesetzt.

Beispiel: Cursor auf Spalte 3, Zeile 8 setzen

```
poke 202,2
poke 205,7
sys 55464
```

Diese Befehle können im Direktmodus eingegeben werden, aber auch in Programmen verwendet werden.

Wie Sie an dem Beispiel sehen, müssen die gewünschten Werte minus eins "gepokt" werden, da für den PLUS/4 intern die oberste Bildschirmzeile nicht die Zeile eins, sondern die Zeile null ist, und die erste Bildschirmspalte nicht eins, sondern ebenfalls null ist. Da die Numerierung des PLUS/4 ab null beginnt, können sich die Spaltenwerte im Bereich zwischen null und 39 und die Zeilenwerte im Bereich zwischen null und 24 bewegen.

5.2. Dauerfunktion für alle Tasten

Wenn Sie wünschen, daß alle Tasten Dauerfunktion besitzen, nicht nur die Editier- und die Leertaste, erreichen Sie das mit:

```
poke 1344,128
```

Auch dieser Befehl kann sowohl im Direktmodus wie in Programmen verwendet werden. Die Wirkung läßt sich rückgängig machen durch:

```
poke 1344,0
```

5.3. Insert-Modus löschen

Was unter dem Insert-Modus zu verstehen ist, wissen Sie alle. Es dürfte wohl kaum einen PLUS/4-Besitzer geben, der sich nicht bereits über seltsame, ungewollt auftretende Steuerzeichen ärgerte.

Sie kommen auf folgende Weise zustande: Bei jedem Druck auf die Taste "Inst" erhöht der PLUS/4 einen Zähler in der Speicherstelle 207, der ihm Auskunft über die sogenannte "Anzahl ausstehender Inserts" gibt. Wenn dieser Zähler ungleich null ist, erwartet der PLUS/4 die Eingabe eines alphanumerischen Zeichens, das eingefügt werden soll und für das zuvor mit der Taste "Inst" Platz geschaffen wurde.

Wird hingegegen eine Editiertaste betätigt (Cursortasten, "Del", "Clear", "Home") erscheint eines der erwähnten - invers dargestellten - Steuerzeichen.

Äußerst unangenehm ist diese Eigenschaft in Programmen mit Editierfunktionen, zum Beispiel zur Eingabe von Datensätzen oder Texten. In professionellen Programmen sollte das Erscheinen dieser ärgerlichen Steuerzeichen unbedingt verhindert werden. Dies ist möglich, wenn vor jeder Ausgabe eines Zeichens der erwähnte Zähler auf null gesetzt wird, dem PLUS/4 somit unabhängig davon, wie oft zuvor "Inst" gedrückt wurde, simuliert wird, daß die Anzahl ausstehender Inserts gleich null ist.

Beispiel:

```
10 getkey a$
20 poke 207,0
30 print a$;
40 goto 10
```

Dieses kleine Programm stellt eine Eingabeschleife dar. Es wartet auf einen Tastendruck und gibt das jeweilige Zeichen aus, nachdem die Anzahl ausstehender Inserts zuvor auf null gesetzt wurde (Zeile 20). Geben Sie das Programm ein, starten Sie es und versuchen Sie, Steuerzeichen bewußt zu erzeugen. Es wird Ihnen nicht gelingen.

Die Anwendung dieses Tricks ist nur in Programmen möglich, nicht im Direktmodus.

5.4. Cursornachlauf verhindern

Eine manchmal sehr störende Eigenschaft des PLUS/4 ist der sogenannte "Tastaturpuffer", in dem alle eingegebenen Zeichen gespeichert werden (bis er "überläuft"!). Wenn ein Programm nicht schnell genug ist, die Eingaben des Benutzers augenblicklich zu verarbeiten, gehen diese nicht verloren. Noch nicht bearbeitete Tastendrucke werden aus dem Tastaturpuffer geholt, in dem sie vor der Verarbeitung gespeichert wurden.

In einer Textverarbeitung müssen nach jedem einzelnen Tastendruck umfangreiche Berechnungen und Ausgaben stattfinden, währendessen der Benutzer bereits weitere Tasten betätigt. Bei einer langsamen Textverarbeitung macht sich der Tastaturpuffer bemerkbar, indem (bei hoher Eingabegeschwindigkeit des Benutzers) die vom Benutzer eingegeben Tasten nachträglich erscheinen können, wenn dieser sich bereits bequem in seinem Sessel zurückgelehnt hat.

Dieses "Nachlaufen" des Cursors kann verhindert werden, wenn dem PLUS/4 vorgeschwindelt wird, daß sich zu keinem Zeitpunkt noch nicht verarbeitete Zeichen im Tastaturpuffer befinden. Zu diesem Zweck muß der "Tastaturpufferzähler" (Speicherstelle 239), in der sich die Anzahl der gespeicherten Zeichen befindet, vor jeder Eingabe auf null gesetzt werden. Nachteil dieser Methode: Nicht verarbeitete Zeichen gehen verloren (eben dies soll der Tastaturpuffer verhindern!).

Beispiel:

```
10 poke 239,0
20 getkey a$
30 print a$;
40 for i=1 to 3000:next
50 goto 10
```

Aufgrund der Warteschleife zwischen jeder Aus- beziehungsweise Eingabe ist dieses Programm viel zu langsam, um schnelle Eingaben in "Realtime" (Echtzeit) zu verarbeiten. Dadurch, daß vor jeder Eingabe der Tastaturpufferzähler auf null gesetzt wird, werden nicht verarbeitete Zeichen einfach gelöscht. Sie werden es durch brutales Bearbeiten der Tastaur kaum schaffen, eine nachlaufende Zeichenausgabe zu erhalten.

Die bisher erwähnten Anwendungen für diesen Trick waren mehr ästhetischer Natur (ein nachhinkender Cursor ist einfach unschön!). Es gibt jedoch Fälle, in denen nicht bearbeitete Zeichen unbedingt gelöscht werden müssen. Lesen Sie hierzu bitte das Kapitel "Reaktionstest". Der Reaktionstest wäre ohne diesen Trick nicht möglich!

5.5. Cursorsimulation im Programmodus

Im Direktmodus sehen Sie zu jedem Zeitpunkt den Cursor auf dem Bildschirm. Im Programmodus leider nur, wenn für Eingaben der "INPUT"-Befehl verwendet wird, der - siehe "Eingabe-Routine" - leider manchen Nachteil besitzt.

Professionelle Eingaberoutinen werden immer mit dem Befehl "GET" programmiert, wobei der Cursor jedoch - wie im Programmodus üblich - ausgeschaltet ist. Ein einfacher - nicht blinkender - Cursor kann simuliert werden, indem das Zeichen an der momentanen Cursorposition invertiert dargestellt wird. Wenn der Cursor bewegt wird, wird die Darstellung des betreffenden Zeichens wieder normalisiert und das Zeichen an der neuen Cursorposition invertiert.

Beispiel:

```
10 cp=3072+peek(202)+40*peek(205)
20 cz=peek(cp)
30 poke cp,cz and 128
40 getkey a$
```

```
50 print a$;
60 poke cp,cz
70 goto 10
```

Um dieses Beispiel zu verstehen, muß man wissen, daß jeder Bildschirmposition eine Speicherstelle im sogenannten "Bildschirmspeicher" des PLUS/4 entspricht, in der das dargestellte Zeichen gespeichert ist.

Die Speicherung erfolgt im sogenannten "Bildschirmcode". In Zeile 10 wird jene Speicherstelle im Bildschirmspeicher "cp" ermittelt, die der momentanen Cursorposition entspricht (cp=Anfangsadresse des Bildschirmspeichers plus Spalte plus 40 mal Cursorzeile).

"cz" ist der Code des Zeichens an der betreffenden Position. Dieser Code (diese Zahl) wird mit "OR 128" verknüpft. Der Wert, der sich durch diese Verknüpfung ergibt, entspricht der reversen Darstellung des gleichen Zeichens.

Dieser Wert wird in die entsprechende Speicherstelle "gepokt". Das Zeichen an der aktuellen Cursorposition wird dadurch invers dargestellt.

Nun wird auf eine Eingabe des Benutzers gewartet. Bevor das betreffende Zeichen ausgegeben und dadurch der Cursor bewegt wird, wird das invertierte Zeichen wieder normalisiert, indem der ursprüngliche Zeichencode "cz" in die ermittelte Speicherstelle "gepokt" wird.

5.6. Simulierter Direktmodus

Viele Möglichkeiten, die der PLUS/4 bietet, lassen sich nicht in Programmen anwenden, zum Beispiel das Löschen, Ändern oder die Neueingabe einer Zeile. Wie nützlich diese Möglichkeiten jedoch sein können, sehen Sie anhand des Programms "REM-Killer".

Mit dem simulierten Direktmodus können alle normalerweise in Programmen nicht nutzbare Fähigkeiten des PLUS/4 auch im Programmodus verwendet werden.

Er beruht auf folgendem Prinzip: Der gewünschte Befehl wird auf den Bildschirm geschrieben. In den Tastaturpuffer werden die ASCII-Codes zur Cursorsteuerung geschrieben, und zwar so, daß der Cursor dadurch exakt auf die Befehlszeile gesetzt wird. Anschließend wird der Code 13 (ASCII-Code von "RETURN") in den Tastaturpuffer geschrieben und der Tastaturpufferzähler mit jener Zahl versehen, die der Anzahl der in den Tastaturpuffer geschriebenen Zeichen entspricht.

Der nächste Programmbefehl ist "END", so unglaublich dies auch klingen mag. Der Grund: Wird ein Programm mit "END" beendet, arbeitet der BASIC-Interpreter des PLUS/4 anschließend alle noch nicht verarbeiteten Zeichen ab, die im Tastaturpuffer enthalten sind und die er nun - nach Beenden des Programms - für Benutzereingaben im Direktmodus hält.

Beispiel:

```
100 scnclr
110 print "delete 120-140:run 200"
120 poke 1319,19:rem 'home'
130 poke 1320,13:rem 'return'
140 poke 239,2:rem tast.p.zaehler
150 end
160 :
200 print "die zeilen 120-140 wurden geloescht"
```

Vor der Programmerläuterung: Der Tastaturpuffer des PLUS/4 beginnt ab Speicherstelle 1319; der Tastaturpufferzähler ist in der Speicherstelle 239 enthalten.

Nachdem der Bildschirm gelöscht wurde, wird in die oberste Zeile der Befehl geschrieben "delete 120-140:run 200" (Zeile

100-110). Dieser Befehl soll im simulierten Direktmodus ausgeführt werden.

In die erste Speicherstelle des Tastaturpuffers wird der Code für "Cursor home" "gepokt" (siehe ASCII-Tabelle im Handbuch), in die zweite Speicherstelle der Code für "RETURN" (Zeile 120-130).

Gemäß der Anzahl "gepokter" Zeichen wird der Tastaturpufferzähler mit dem Wert zwei versehen (Zeile 140), bevor der Befehl "END" folgt.

Der BASIC-Interpreter arbeitet nun die im Tastaturpuffer vorhandenen Zeichen ab und führt sie aus, als wären die zugehörigen Tasten "Home" und "RETURN" soeben - nach Beenden des Programms - vom Benutzer betätigt worden.

Der Cursor wird in die oberste Bildschirmzeile gesetzt ("Home"), in der sich die gewünschten Befehle befinden, und diese ausgeführt ("Return"). Dadurch werden die Zeilen 120-140 gelöscht und das Programm ab Zeile 200 neu gestartet. Der Kommentar "die zeilen 120-140 wurden geloescht" wird ausgegeben. Anschließend wird das Programm endgültig beendet.

Merken Sie sich bei Anwendung dieser Methode bitte, daß unbedingt ein Startbefehl (im Beispiel "RUN 200") auf den Bildschirm geschrieben und anschließend im Direktmodus ausgeführt werden muß. Nur auf diese Weise ist es möglich, den Programmodus vorübergehend zu verlassen, um einen Direktmodusbefehl auszuführen, und danach im Programmodus weiterzuarbeiten.

Wichtig ist, daß sowohl durch den "DELETE"-Befehl als auch durch das Neustarten mit "RUN 200" alle Variablen gelöscht werden!

Die Inhalte von im weiteren Programmlauf benötigten Variablen müssen daher in (freie!) Speicherstellen "gepokt" werden, um nach dem Neustart des Programms wieder aus diesen Speicherstellen gelesen zu werden (siehe "Rem-Killer").

Solche freien Speicherstellen bietet zum Beispiel der Bereich 1015 bis 1078.

6. Programmierung des PLUS/4 in Maschinensprache

Die folgende Einführung in die Programmierung des PLUS/4 in Maschinensprache kann Ihnen leider nur in sehr geringem Maße Kenntnisse über die Befehle dieser höchst effizienten und extrem schnellen (und umständlichen!) Programmiersprache vermitteln. Im folgenden wird vorwiegend dargestellt, welche Besonderheiten beim Einsatz dieser Sprache auf dem PLUS/4 zu beachten sind. Zum Erwerb allgemeiner Maschinensprachekenntnisse empfehle ich Ihnen die Lektüre entsprechender Bücher für den verbreiteteren Bruder des PLUS/4, den C64, und das Studium von Fachzeitschriften.

6.1. Von BASIC zu Assembler

Bevor ich auf die Besonderheiten der Programmierung des PLUS/4 in Maschinensprache eingehe, will ich kurz erläutern, welche Vorteile diese Programmiersprache gegenüber BASIC bietet und warum sie jedem ambitionierten Programmierer als Alternative zu BASIC empfohlen werden kann.

BASIC ist eine sogenannte "höhere Programmiersprache", die ein komfortables Programmieren erlaubt, für Ihren PLUS/4 jedoch im Grunde unverständlich ist. Das Herz eines Heimcomputers bildet der sogenannte "Mikroprozessor". Jeder Mikroprozessor besitzt seinen eigenen Befehlsvorrat, seine eigene Sprache.

Diese Sprache ist leider äußerst unanschaulich und schwer zu verstehen. Daher wurden die höheren Programmiersprachen entwickelt, von denen BASIC die größte Verbreitung gefunden hat, da auch Computerneulinge diese Sprache sehr schnell erlernen und damit umgehen können.

Der im PLUS/4 eingebaute Mikroprozessor versteht leider nicht einen der Befehle, die BASIC bietet. Daher besitzt der PLUS/4 einen integrierten sogenannten "BASIC-Interpreter". Dieser

Interpreter ist ein Programm, das selbst in Maschinensprache geschrieben ist und die Aufgabe hat, jeden einzelnen BASIC-Befehle in die Sprache des Mikroprozessors zu übersetzen.

Diese Übersetzung geschieht unbemerkt von Ihnen ständig in einem laufenden BASIC-Programm. Der gesamte Vorgang ist leider hochkompliziert und benötigt dementsprechend viel Zeit.

BASIC-Programme sind daher ungleich langsamer als Programme, die in Maschinensprache geschrieben werden und daher - ohne Übersetzungsvorgang - direkt vom Mikroprozessor verstanden werden.

Für die meisten Anwendungen ist die Geschwindigkeit eines BASIC-Programms vollkommen ausreichend. Besonders zeitkritische Programme erfordern jedoch den Einsatz von Maschinensprache. Meist genügt es, nicht das komplette Programm, sondern ausschließlich zeitkritische Programmteile in Maschinensprache zu schreiben.

Exakt diese Vorgehensweise, Unterprogramme in Maschinensprache als Ersatz für zeitkritische BASIC-Routinen zu verwenden, wird in mehreren in diesem Buch vorgestellten Programmen verwendet.

Im folgenden wird erläutert, wie die Einbindung solcher Maschinenroutinen in BASIC-Programme auf dem PLUS/4 vorgenommen wird.

Zuvor will ich Ihnen jedoch die enormen Vorteile der Maschinensprache an einem Beispiel demonstrieren. Stellen Sie sich vor, Sie wollen den gesamten Bildschirm mit dem Buchstaben "A" füllen. In BASIC könnte diese Aufgabe mit folgendem Programm gelöst werden:

```
10 scnclr
20 for i=1 to 1000
30 print "a";
40 next
```

Der Bildschirm wird gelöscht, anschließend wird 1000mal das Zeichen "A" ausgegeben, bis der gesamte Bildschirm damit gefüllt wurde.

Da der Mikroprozessor des PLUS/4 keinen der verwendeten BASIC-Befehle versteht, werden diese vom eingebauten BASIC-Interpreter während des Programmablaufs in Maschinenbefehle übersetzt.

Als Konsequenz ergibt sich ein äußerst langsames Programm, das mehrere Sekunden benötigt, um die 1000 Zeichen auszugeben. Hinter der Zeichenausgabe steht folgender Vorgang: Den 1000 Bildschirmpositionen entsprechen 1000 Speicherstellen im PLUS/4, der sogenannte "Bildschirmspeicher". Dieser Bereich beginnt ab Speicherstelle Nummer 3072 (jede Speicherstelle im PLUS/4 besitzt eine eigenen "Hausnummer", über die sie angesprochen, "adressiert" werden kann).

Die Speicherstelle 3072 entspricht dem ersten Zeichen in der linken oberen Bildschirmecke (Spalte null in Zeile null). Die nächste Speicherstelle mit der Nummer 3073 entspricht dem Zeichen in Spalte eins von Zeile null und so weiter.

Jede dieser Speicherstellen kann eine Zahl zwischen null und 255 enthalten, wobei jede Zahl einem anderen Zeichen zugeordnet ist. Dem Zeichen "A" ist die Zahl eins, dem Zeichen "B" die Zahl zwei zugeordnet. Sie können dies selbst ausprobieren, indem Sie die Speicherstelle 3072 mit dem Wert eins versehen:

```
poke 3072,1
```

In der linken oberen Ecke wird das Zeichen "A" erscheinen, das heißt in der der Speicherstelle 3072 zugeordneten Bildschirmposition erscheint jenes Zeichen, das der Zahl eins zugeordnet ist.

Auf diese Weise könnte man nun den gesamten Bildschirm mit dem Zeichen "A" füllen:

```
10 for i=3072 to 3072+999
20 poke i,1:rem code fuer "a"
30 next
```

Dieses Programm erfüllt zwar die ihm zugedachte Funktion, zeichnet sich jedoch leider ebenfalls durch eine sehr geringe Ablaufgeschwindigkeit aus. Der Grund liegt darin, daß es sich zwar um ein "machinennahes" Programm handelt, jedoch immer noch um ein BASIC-Programm, das während des Programmablaufs Befehl für Befehl übersetzt werden muß.

Die Vorgehensweise selbst, das Füllen bestimmter Speicherstellen mit bestimmten Zahlen, ist jedoch problemlos in ein echtes Maschinenspracheprogramm übersetzbar. Das entsprechende Maschinenspracheprogramm:

```
lda #$00
sta $0c00
sta $0c01
sta $0c02
...
...
...
```

Dieses Programm besteht aus Maschinensprachebefehlen, die vom Rechner unmittelbar verstanden werden und kann mit dem im PLUS/4 eingebauten "Monitorprogramm" eingegeben werden (siehe "Der integrierte Monitor").

Die verwendeten Befehle sind zweifellos sehr unverständlich für Sie, unter anderem auch aufgrund der "hexadezimalen" Eingabe anstelle der gewohnten dezimalen Schreibweise.

Der Befehl "lda #$01" versieht den sogenannten "Akkumulator" mit dem Wert eins. Stellen Sie sich unter dem Akkumulator am besten eine spezielle Speicherstelle vor, die gegenüber den restlichen Speicherstellen den Vorteil vielseitigerer Verwendungsmöglichkeiten besitzen.

Der Befehl "sta $0c00" überträgt Inhalt des Akkumulators (eins) in die Speicherstelle $0c00. $0c00 entspricht der Dezimalzahl 3072, das heißt der ersten Speicherstelle des Bildschirmspeichers, die somit mit der Zahl eins gefüllt wird.

Der folgende Befehl "sta $0c01" überträgt den Inhalt den Akkumulators, der nicht verändert wurde, in die Speicherstelle $0c01 (=dezimal 3073) und damit in die nächste Speicherstelle des Bildschirmspeichers.

Der Befehl "sta $0c02" versieht entsprechend jene Speicherstelle mit dem Wert eins, dem Code des Zeichens "A", die der Bildschirmposition Spalte zwei in Zeile null entspricht.

Die Programmierung in Maschinensprache besteht vorwiegend aus Befehlen, die bestimmte spezielle Speicherstellen (Akkumulator) mit bestimmten Inhalten versehen, und einer weiteren Befehlsklasse, die es ermöglicht, den Inhalt dieser Speicherstellen in beliebige andere zu übertragen.

Da es ein wenig aufwendig wäre, zum Beschreiben des Bildschirmspeichers ein Maschinensprachprogramm zu schreiben, das aus 1000 einzelnen Übertragungsbefehlen ("sta $....") besteht, verwendet man in der Praxis ebenso wie in BASIC-Programmen Schleifenstrukturen. Mit Hilfe dieser Schleifen läßt sich die vorliegende Aufgabe erheblich kürzer lösen:

```
a 065e  lda #$01
a 0660  ldx #$00
a 0662  sta $0c00,x
a 0664  dex
a 0665  bne $0662
a 0667  rts
```

Dieses recht kurze Programm schreibt exakt 256mal das Zeichen "A" auf den Bildschirm und füllt daher bereits ein Viertel des kompletten Bildschirms mit diesem Zeichen, analog dem BASIC-Programm:

```
10 for i=3072 to 3072+255
20 poke i,a
30 next
```

Dieses Maschinenprogramm können Sie mit dem eingebauten "Monitorprogramm" des PLUS/4 eingeben. Die Beschreibung dieses sogenannten "Monitors", der ein unverzichtbares Hilfsmittel für die Programmierung des PLUS/4 in Maschinensprache darstellt, geschieht im nachfolgenden Kapitel.

Wenn Sie das Programm eingeben und mit dem Befehl "sys 1630" starten, werden Sie feststellen, daß - im Gegensatz zum entsprechenden BASIC-Programm - der Bildschirm praktisch sofort mit dem Zeichen "A" beschrieben wird. Das Maschinenspracheprogramm ist mehrere hundert Mal schneller als ein BASIC-Programm mit der gleichen Funktion!

Soviel zum Sinn und der Anwendung von Maschinensprache. Sollte Ihr Interesse an dieser extrem schnellen Programmiersprache geweckt sein: Die Befehle dieser Sprache sind keineswegs so einheitlich wie die Befehle einer höheren Programmiersprache und können von Rechner zu Rechner völlig unterschiedlich sein, je nach eingebautem Mikroprozessor und dessen speziellem Befehlssatz.

Der PLUS/4 besitzt jedoch glücklicherweise einen Mikroprozessor, der die gleiche Sprache spricht wie die sehr häufig verwendeten Typen 6502 und 6510, die zum Beispiel im Commodore 64 oder im Commodore 128 verwendet werden.

Zum Erlernen der Maschinensprache, die auf dem PLUS/4 eingesetzt wird, ist daher jegliche Literatur geeignet, die sich mit der Programmierung der Mikroprozessoren 6502 oder 6510 befasst. Ebenso ist jegliche Literatur geeignet, die in die Maschinenspracheprogrammierung des C64 oder des C128 einführt.

Nicht übertragbar sind hingegen die verwendeten Adresssen, da zum Beispiel der Bildschirmspeicher des C64 nicht mit Speicherstelle Nummer 3072, sondern mit der Nummer 1024 beginnt. Dies ändert jedoch nichts an den geschilderten Prinzipien der Programmierung und den verwendeten Befehlen.

In den folgenden Kapiteln werde ich erläutern, wie der eingebaute Monitor zu bedienen ist, und die Adressen der wichtigsten PLUS/4-spezifischen Speicherbereiche angeben.

6.2. Der integrierte Monitor

Üblicherweise wird nicht in der eigentlichen Maschinensprache programmiert, sondern in "Assembler". Zur Assemblerprogrammierung wird jedoch ein Hilfsprogramm benötigt, das gleichfalls den Namen Assembler trägt. Ein Mini-Assembler ist in dem Monitorprogramm des PLUS/4 vorhanden. Dieser Monitor besitzt weitere Möglichkeiten außer dem "Assemblieren", das heißt dem Umwandeln der Assemblerbefehle in Maschinensprachebefehle.

Im Handbuch zum PLUS/4 wird die Existenz dieses Monitors zwar erwähnt, seine Bedienung jedoch nicht im geringsten erläutert. Sollte Ihnen die folgende Kurzanleitung nicht umfassend genug sein, verweise ich wiederum auf die entsprechende Literatur (zum C64). Glücklicherweise sind die meisten Monitorbefehle und ihre Syntax standardisiert, es gibt nur geringe Unterschiede in der Bedienung verschiedener Monitore.

Aufruf des Monitors:

Zum Aufruf genügt die Eingabe des BASIC-Befehls "MONITOR". Der Monitor meldet sich mit der Anzeige der aktuellen Registerinhalte und des Programmcounters. Alle im folgenden genannten Monitorbefehle verlangen hexadezimale Eingaben! Bei den Befehlen "M ..." und "D..." genügt die Eingabe von "M" beziehungsweise "D", um die weitere Ausgabe ab der aktuellen Adresse zu veranlassen.

Inhalte von Speicherstellen ausgeben:

Die Inhalte beliebiger Speicherbereiche können Sie mit dem Befehl "M (ADR1) (ADR2)" in hexadezimaler Form ausgeben lassen.

Beispiel: "M 0200 0300" führt zur Ausgabe der Inhalte der Speicherstellen $0200-$0400.

Inhalte von Speicherstellen ändern:

Lassen Sie sich den Inhalt der gewünschten Speicherstelle(n) ausgeben, bewegen Sie den Cursor zum Zeilenanfang, überschreiben Sie den Punkt mit dem Zeichen ">", ändern Sie die ausgegebenen Werte und bestätigen Sie die Änderung mit "Return". Selbstverständlich können Sie auch direkt das Zeichen ">" eingeben und dahinter eine beliebige Byte-Folge.

Beispiel: "> 0200 02 4A 2B

Disassemblieren:

Ein beliebiger Speicherbereich wird mit dem Befehl "D (ADR1) (ADR2)" disassembliert ausgegeben, das heißt anstelle der Zahlencodes erhalten Sie die zugehörigen Assemblerbefehle. Das Disassemblieren ist nur sinnvoll in einem Bereich, der tatsächlich ein Programm enthält.

Beispiel: "D 1000 1100" gibt den Bereich $1000-$1100 disassembliert aus.

Assemblieren:

Zum Assemblieren benötigt der Monitor die Startadresse, ab der assembliert werden soll und die Eingabe des ersten Assemblerbefehls. Der Befehl zum Assemblieren lautet "A". Nachdem Sie den ersten Assemblerbefehl eingegeben haben, wird Ihnen die nächste Adresse automatisch vorgegeben, sie können den zweiten Assemblerbefehl sofort eingeben. Wichtig ist nur der korrekte Start des Assembliervorgangs. Beachten Sie, daß die Befehle in hexadezimaler Form eingegeben werden und vor einer Zahl oder Adresse das Zeichen "$" eingegeben werden muß!

Beispiel:

```
"A $4000 LDA #$01"
". $4002 LDX $FF
". $4004 JSR $5000"
". $4007 RTS"
```

Speicherbereiche kopieren:

Mit dem Befehl "T (ADR1) (ADR2) (ADR3)" kann ein beliebiger Speicherbereich kopiert werden.

Beispiel: "T 1000 2000 3000" kopiert die Inhalte der Speicherstellen $1000-$2000 nach $3000-$4000.

Bereiche abspeichern:

Zweifellos ist dies einer der wichtigsten Befehle. Was nützt das schönste Maschinenspracheprogramm, wenn es nicht abge-

speichert werden kann. Der zugehörige Befehl hat die Syntax 'S"(NAME)", (GERÄT), (ANFANGSADRESSE), (ENDADRESSE +1)'.

Beispiel: 'S"TEST",08,1000,2001' speichert den Bereich $1000-$2000 unter dem Namen "TEST" auf Diskette.

Programm laden:

Dieser Befehl hat die Syntax 'L"(NAME)",(GERÄT).

Beispiel: 'L"TEST",08' lädt das Programm "TEST" von Diskette an jene Adresse, an der es sich vor dem Abspeichern befand.

Monitor verlassen:

Durch die Eingabe von "X" wird der Monitor verlassen.

Dies waren zwar bei weitem nicht alle, jedoch die wichtigsten Befehle, über die der integrierte Monitor verfügt. Ein Tip zum Abschluß: Oftmals möchte man eine Ausgabe (in hexadezimaler Form oder dissasembliert) statt auf dem Bildschirm auf einem Drucker ausgeben lassen. Dies ist möglich, obwohl der Monitor keinen speziellen Befehl besitzt. Die Vorgehensweise:

1. Eingabe von "OPEN 4,4:CMD 4"
2. Monitor aufrufen mit "MONITOR"
3. Befehl eingeben, zum Beispiel "D 1000 1100"
4. Monitor verlassen mit "X"
5. Ausgabe beenden mit "PRINT#4:CLOSE 4"

Im ersten Schritt wird ein Kanal zum Drucker geöffnet und alle weiteren Ausgaben werden auf diesen Kanal "gelegt", auch die folgenden Monitorausgaben werden nun auf dem Drucker ausgegeben.

Sie verfügen nun über das notwendige "Handwerkszeug" zur Programmierung des PLUS/4 in Maschinensprache. Vielleicht wird es eines Tages auch einen vernünftigen Assembler geben (der in Monitoren integrierte Assembler ist nur sehr bedingt für umfangreichere Programme geeignet), so daß auch große Text- oder Dateiverwaltungsprogramme für den PLUS/4 vollständig in Assembler geschrieben werden können. Momentan muß hierzu noch der Umweg über einen anderen Rechner genommen werden.

6.3. Einbindung von Maschinenroutinen

Sie sind zwar nun in der Lage Assemblerprogramme zu schreiben, sie wissen jedoch noch nicht, wohin Sie diese legen sollen. Mehrere Bereiche können für kürzere Routinen verwendet werden. Da diese Bereiche jedoch alle für bestimmte Zwecke gedacht sind, bestehen jeweils verschiedene Einschränkungen, die beachtet werden müssen.

Der Eingabepuffer/BASIC-Puffer ($0200-$025C):

Dieser Bereich reicht von $0200-$025C. Da er vom BASIC-Interpreter verwendet wird, darf er jedoch ausschließlich für reine Maschinenprogramme benutzt werden, nicht in Verbindung mit einem BASIC-Programm!

Der RS-232-Puffer ($03F7-$0436):

Maschinenroutinen, die in diesem Bereich abgelegt werden, sind vor dem Überschreiben geschützt, wenn die RS-232-Schnittstelle nicht verwendet wird (zum Beispiel zum Betrieb eines Modems).

Der Funktionstastenspeicher ($055F-$05E6):

In diesem Bereich wird die Belegung der Funktionstasten gespeichert, das heißt die zugehörigen Zeichenketten. Dieser Be-

reich darf ohne Einschränkungen verwendet werden, solange die Funktionstastenbelegung nicht umdefiniert wird. In diesem Fall wird das Maschinenprogramm sofort durch die jeweilige Zeichenkette überschrieben.

Der Sprachsynthesizer ($065E-$06EB):

Dieser Bereich wird nur dann überschrieben, wenn der Sprachsynthesizer verwendet wird. Solange die Sprachausgabe nicht verwendet wird, sind Maschinenroutinen vor dem Überschreiben geschützt.

Verschieben des BASIC-Bereichs:

Sollte Ihr Platzbedarf jedoch größer sein, müssen Sie Teile des BASIC-Speichers verwenden. Hierzu muß der BASIC-Anfang verschoben werden. Lassen Sie sich den Wert von Speicherstelle 44 ausgeben ("PRINT PEEK(44)"), in der sich der "High"-Teil des Zeigers auf den BASIC-Anfang befindet. Dieser Wert beträgt normalerweise 16, wovon ich im folgenden ausgehen werde. "Poken" Sie in diese Speicherstelle den Wert 16 plus vier mal die Anzahl der Kilobyte, die Sie für Ihr Maschinenprogramm benötigen.

Geben Sie anschließend ein: `"POKE PEEK(44)*256,0:NEW"`.

Der BASIC-Anfang wurde durch diese Befehle um die gewünschte Kilobytezahl verschoben. Ihr Maschinenprogramm darf nun von $1000 (dezimal 4096), dem bisherigen BASIC-Beginn, bis zum geänderten BASIC-Anfang reichen.

BASIC-Anfang um 1 kB verschieben:

```
1. POKE 44,16+4
2. POKE PEEK(44)*256,0
3. NEW
```

6.4. Die wichtigsten Zeropage-Adressen

Da ich nun davon ausgehen kann, daß Sie Grundkenntnisse in der Assemblerprogrammierung besitzen, wissen Sie, wie wichtig die Zeropage eines Rechners ist. Ohne sie ist beim PLUS/4 keine indizierte Adressierung möglich, da bei dieser Adressierungsform Zwei-Byte-Zeiger in der Zeropage abgelegt werden müssen. Außerdem verlaufen Zugriffe auf die Zeropage sehr schnell und die Maschinensprachebefehle sind um je ein Byte kürzer als es bei Verwendung von 16-Bit-Adressen der Fall wäre.

Mehrere Bereiche der Zeropage dürfen Sie - wieder mit Einschränkungen - für Zeiger, Variablen und Tabellen verwenden.

Der Anwenderbereich ($D8-$E8):

Dieser Bereich ist völlig unbenutzt und wird speziell für Anwenderprogramme zur Verfügung gestellt, was ich als sehr nette Geste empfinde.

Die Fließkommaakkumulatoren ($61-$70):

In den beiden Fließkommaakkumulatoren ("FAC1" und "FAC2") werden vom BASIC-Interpreter Berechnungen vorgenommen. Maschinenroutinen, die in Verbindung mit BASIC-Programmen als Unterprogramme eingesetzt werden, dürfen daher keine permanent benötigten Daten in diesem Bereich speichern, da diese nach der Rückkehr in das BASIC-Programm verlorengehen.

Der Sprachsynthesizer-Bereich ($D0-$D7):

Dieser Bereich ist der Arbeitsbereich für den Sprachsynthesizer und kann bedenkenlos verwendet werden, solange dieser nicht benutzt wird.

6.5. Betriebssystem-Routinen

Wenn Sie in Assembler beispielsweise eine komfortable Eingaberoutine als "INPUT"-Ersatz schreiben wollen, sind Sie "aufgeschmissen" ohne Kenntnisse über die wichtigsten Betriebssystemroutinen. Wie wollen Sie zum Beispiel die Tastatur abfragen?

Es gibt Unmengen von Betriebssytemroutinen (alle sind in Unterprogrammform geschrieben), von denen ich im folgenden nur die wichtigsten und ständig benötigten aufzählen werde.

BSOUT ($FFD2)

Zur Ausgabe eines Zeichens auf dem Bildschirm können Sie dieses in die entsprechende Adresse des Bildschirmspeichers "poken". Komfortabler wird die Zeichenausgabe jedoch bei Verwendung der Routine "BSOUT", die ein im Akku übergebenes Zeichen an der momentanen Cursorposition ausgibt. Der Akku muß mit dem ASCII-Code des auszugebenden Zeichens geladen werden. Es können beliebige Zeichen ausgegeben werden, auch Cursorsteuerzeichen, "Inst", "Del" etc.

Ausgabe von "A" (ASCII-Code ~~64=$40~~ 65=$41):

```
1. "LDA #$40"
2. "JSR BSOUT"
```

GETIN ($FFE4)

"GETIN" holt ein Zeichen aus dem Tastaturpuffer, das heißt diese Routine fragt die Tastatur ab. Ein eingegebenes Zeichen wird im Akku übergeben (im ASCII-Code). "GETIN" wartet jedoch, bis tatsächlich eine Taste gedrückt wurde, arbeitet also analog zu "GET" und nicht zu "GETKEY". Wurde keine Taste betätigt, übergibt "GETIN" im Akku eine null und das Zeroflag ist gesetzt.

Auf eine Taste warten/Zeichen ausgeben:

```
1. "LABEL  JSR $FFE4"
2. "       BEQ LABEL"
3. "       JSR $FFD2"
```

PLOT ($FFF0)

Mit "PLOT" kann der Cursor auf eine beliebige Position gesetzt werden. Im Y-Register wird die Spalte (0-39) und im X-Register die Zeile (0-24) übergeben. Das Carryflag muß gelöscht sein.

Cursor auf Spalte zwei von Zeile drei setzen:

```
1. "LDY #$01"
2. "LDX #$02"
3. "CLC
4. "JSR $FFF0"
```

Beachten Sie, daß die erste Zeile die Nummer null besitzt. Entsprechendes gilt für die Spalte. Wenn das Carryflag beim Aufruf von "PLOT" gesetzt ist, wird die aktuelle Cursorposition im X- beziehungsweise im Y-Register übergeben.

6.6. Routinen des BASIC-Interpreters

Der BASIC-Interpreter besitzt einige äußerst nützliche Routinen, mit denen zum Beispiel aus einem BASIC-Programm heraus Assemblerroutinen aufgerufen und ihnen zugleich Parameter übergeben werden können. Drei der wichtigsten Routinen sind "CHKKOM", "GETBYT" und "AXOUT".

CHKKOM ($9491)

"CHKKOM" liest ein Zeichen aus dem BASIC-Text und prüft, ob es sich um ein Komma handelt. Wenn dies nicht der Fall ist, wird ein "syntax error" ausgegeben. Diese Routine kann in Verbindung mit "GETBYT" sinnvoll eingesetzt werden.

GETBYT ($9D84)

"GETBYT" liest einen Ein-Byte-Wert aus dem BASIC-Text, das heißt eine Zahl zwischen null und 255. Die betreffende Zahl wird im X-Register übergeben.

"CHKKOM" und "GETBYT" werden üblicherweise gemeinsam eingesetzt, um Parameter aus dem BASIC-Text zu lesen. Ein Beispiel: Sie schreiben eine Eingaberoutine in Assembler, die vom BASIC-Hauptprogramm aufgerufen wird. Dieser Eingaberoutine sollen Spalte und Zeile übergeben werden, an der die Eingabe beginnen soll:

```
1. "ADRESSE JSR $9491
2. "        JSR $9D84
3. "        STX $D8
4. "        JSR $9491
5. "        JSR $9D84
6. "        RTS
```

Aufgerufen wird die Routine zum Beispiel mit "SYS 2*4096,5,10", wenn Sie sie mit dem Monitor ab Adresse $2000 eingeben. Im ersten Schritt wird das dem "SYS"-Befehl folgende Komma gelesen, danach der Wert fünf, der im X-Register übergeben und anschließend nach $D8 gerettet wird, im vierten Schritt wird das zweite Komma gelesen und danach die folgende zehn im X-Register übergeben.

Alle benötigten Parameter wurden nun übergeben und die eigentliche Eingaberoutine kann ablaufen.

AXOUT ($A45F)

Oftmals ist es nötig, Integerwerte auf dem Bildschirm auszugeben, zum Beispiel in Spielen die momentan erreichte Punktzahl. Nehmen wir an, die Punktzahl betrage 1025 (hexadezimal $0401) und sei in zwei Speicherstellen als Low- und High-Byte abgelegt ($01/$04).

Die Ausgabe dieser Zwei-Byte-Zahl auf dem Bildschirm ist ein sehr interessantes Problem, das zwar auf verschiedene Weise gelöst werden kann, jedoch niemals ohne recht umständliche Programme.

Dieses scheinbar unbedeutende und dennoch komplexe Problem kann mit der Routine "AXOUT" umgangen werden. Diese Routine gibt einen beliebigen Zwei-Byte Wert - der im Akku und im X-Register übergeben werden muß - auf dem Bildschirm ab der momentanen Cursorposition aus. Vor dem Aufruf der Routine muß das X-Register mit dem Low- und der Akku mit dem High-Byte der betreffenden Zahl geladen werden.

Ausgabe der Zahl 1025 ($0401)

```
1. "(ADRESSE) LDX #$01"
2. "          LDA #$04"
3. "          JSR $A$5F
4. "          RTS"
```

Geben Sie dieses Programm mit dem Monitor zum Beispiel ab Adresse $2000 ein und rufen Sie es mit "SYS 2*4096" auf. Wie Sie sehen, wird die Zahl 1025 korrekt ausgegeben.

Und nun wünsche ich Ihnen noch viel Spaß bei der Programmiererung in Assembler und zahlreiche "Abstürze".

7. Tips & Tricks für Fortgeschrittene

Dieses Kapitel richtet sich vorwiegend an Leser, die der Abschnitt über die Programmierung des PLUS/4 in Maschinensprache "auf den Geschmack" gebracht hat. Fast alle der im folgenden zu behandelnden Routinen wurden in Maschinensprache geschrieben (Ausnahme: "Programmgesteuerte Funktionstasten"). Reine BASIC-Programmierer sollten dieses Buch nun nicht gleich beiseite legen. Alle vorzustellenden Routinen können von jedem Leser problemlos auch ohne Kenntnis des Programmablaufs in eigenen Programmen verwendet werden.

7.1. Verbesserte LIST-Routine

Wenn Sie mit der BASIC-Programmierung des PLUS/4 bereits so weit vertraut sind, daß Ihre Programme sich über mehrere Bildschirmseiten erstrecken, werden Sie schnell gewisse Mängel des LIST-Befehls feststellen.

Dieser Befehl arbeitet zwar völlig korrekt und ist sehr flexibel, es ist jedoch außerordentlich störend, daß bei langen Programmen Dutzende einzelner LIST-Befehl einzugeben sind, um einen Überblick über das gesamte Programm zu erhalten.

Die vorgestellte Routine erlaubt Ihnen nach einmaligem Aufruf mit "RUN 10000" das "Blättern" in Ihrem Programmlisting mit Hilfe der Cursortasten, ohne daß auch nur ein weiterer Befehl einzugeben wäre.

Programmbedienung

Wenn Sie das folgende Programmlisting eingegeben haben, starten Sie die LIST-Routine bitte mit "RUN 10000". Auf dem Bildschirm werden die Programmzeilen 100 bis 200 aufgelistet. Mit der Taste "CURSOR RECHTS" können Sie in Hunderterschritten weiterblättern (200-300, 300-400, 400-500 usw.), und mit der Taste "CURSOR LINKS" kann ebenfalls in Hunderter-

schritten im Listing zurückgeblättert werden (300-400, 200-300, 100-200, 0-100).

Die Tasten "CURSOR UNTEN" und "CURSOR OBEN" gestatten es, größere Programmabschnitte schnell zu überspringen. "CURSOR UNTEN" blättert in Tausenderschritten weiter (200-300, 1200-1300, 2200-2300 usw.), "CURSOR OBEN" blättert in Tausenderschritten zurück (1200-1300, 200-300).

Die Routine kann vor alllem dann sinnvoll angewendet werden, wenn Sie Ihre Programme in Zehnerschritten durchnumerieren (10, 20, 30 usw.). In diesem Fall werden maximal elf Programmzeilen aufgelistet, so daß kein "Scrollen" des Bildschirms erfolgt und alle aufgelisteten Programmzeilen auch tatsächlich auf dem Bildschirm zu sehen sind.

Das Programm listet immer einen Hundertzeilen-Bereich, zum Beispiel den Bereich 300 bis 400. Dieser Bereich kann geändert werden, indem der Wert 100 in den Zeilen 10090, 10040 und 10050 entsprechend geändert wird. Wird als neuer Wert zum Beispiel 50 eingesetzt, erfolgt die Auflisting in Fünfzigzeilen-Schritten (100-150, 150-200, 200-250 usw.).

Wenn Sie diese Routine in eigenen Programmen einsetzen wollen, empfiehlt sich folgende Vorgehensweise:

1. Löschen Sie das Demoprogramm mit dem Befehl "DELETE -500".

2. Speichern Sie das verbleibende Hauptprogramm zum Beispiel unter dem Namen "LIST" auf Diskette.

3. Wenn Sie mit einem größeren Programmprojekt beginnen, laden Sie das Programm "LIST" und beginnen Sie mit der Erstellung Ihres Programms. Die LIST-Routine stört hierbei dank der sehr hoch gewählten Zeilennummern nicht.

Auf diese Weise ist die LIST-Routine in jedem Ihrer Programme vorhanden und kann jederzeit mit "RUN 10000" aufgerufen

werden, wenn Sie einen Überblick über Ihr Gesamtprogramm erhalten wollen. Solange Sie die beschriebenen möglichen Änderungen nicht vornehmen, wird die Routine wie folgt gesteuert:

1. Aufruf mit "RUN 10000"

2. Cursor rechts: Listet in Hunderterschritten die nächsten Programmteile

3. Cursor links: Listet in Hunderterschritten die vorigen Programmteile

4. Cursor unten: Listet in Tausenderschritten die nächsten Programmteile

5. Cursor oben: Listet in Tausenderschritten die vorigen Programmteile

6. Mit "STOP" kann das Listen jederzeit unterbrochen werden

Programmlisting

```
100 rem ***********************
110 rem *     listroutine     *
120 rem * (c) s.baloui, 1986 *
130 rem ***********************
140 :
150 :
160 rem * demoprogramm *
170 :
180 rem scrollende listroutine
190 rem scrollende listroutine
200 rem scrollende listroutine
210 rem scrollende listroutine
220 rem scrollende listroutine
230 rem scrollende listroutine
240 rem scrollende listroutine
250 rem scrollende listroutine
```

```
260 rem scrollende listroutine
270 rem scrollende listroutine
280 rem scrollende listroutine
290 rem scrollende listroutine
300 rem scrollende listroutine
310 :
320 :
330 rem *** listroutine ***
10000 k1$="scnclr:list 00000-00000"+chr$(13)+"goto 10030"+chr$(13)
:key 1,k1$
10010 ls=100:goto 10080
10020 :
10030 poke 239,0:getkey a$
10040 if a$=chr$(29) then ls=ls+100
10050 if a$=chr$(157) then ls=ls-100
10060 if a$=chr$(17) then ls=ls+1000
10070 if a$=chr$(145) then ls=ls-1000
10080 :
10090 mid$(k1$,13,5)=right$("     "+str$(ls),5)
10100 mid$(k1$,19,5)=right$("     "+str$(ls+100),5)
10110 key 1,k1$:rem key 1 neu belegen
10120 poke 1373,len(k1$):poke 1374,0:poke 239,1:rem f1 simulieren
10130 end:rem direktmodus
```

Programmablauf

Wie im Kapitel "Simulierter Direktmodus" beschrieben wurde, kann jede Taste normalerweise mit Hilfe des Tastaturpuffers simuliert werden, das heißt es ist möglich, den BASIC-Interpreter des PLUS/4 zu "beschummeln". Das Programm "REM-Killer" nutzte diese Möglichkeit aus, um vom Benutzer eingegebene Befehle zum Löschen von Kommentarzeilen vorzutäuschen.

Diese Methode des simulierten Direktmodus wird jedoch durch die Größe des Tastaturpuffers begrenzt, der von 1319 bis 1328 reicht und daher maximal zehn Zeichen aufnehmen kann.

Längere Benutzereingaben können jedoch mit einem Trick ebenfalls simuliert werden. Der Speicher für die Belegung der

Funktionstasten umfasst 135 Zeichen (1375 bis 1510) und bietet daher selbst für umfangreiche Zeichenketten ausreichend Platz.

Das Problem ist nun jedoch, wie es möglich ist, dem PLUS/4 vorzuschwindeln, eine Funktionstaste würde betätigt, obwohl das nicht der Fall ist. Wenn Sie versuchen, diese simulierte Tastenbetätigung ebenso wie bei jeder anderen Taste ausschließlich mit Hilfe des Tastaturpuffers zu erreichen, werden Sie leider keinen Erfolg haben.

Der PLUS/4 behandelt die Funktionstasten nicht wie eine beliebige andere Taste. Bei gedrückter Funktionstaste enthält der Tastaturpufferzähler (238) zwar ebenso wie bei jeder anderen Taste den Wert eins (zum Zeichen, daß eine (!) noch zu verarbeitende Taste gedrückt wurde), benötigt jedoch zusätzlich noch Informationen in folgenden Speicherzellen:

Speicherzelle 1373 = Länge des Funktionstastentextes
Speicherzelle 1374 = Nummer der Funktionstaste (0-7)

Wurde Funktionstaste F1 beispielsweise mit der Zeichenfolge "LIST 1000-2000" belegt und soll eine Betätigung dieser Taste simuliert werden, muß in 1373 der Wert 14 gepokt werden, die Länge dieser Zeichenkette.

In Speicherzelle 1374 wird nicht der Wert eins, sondern null gepokt, da der PLUS/4 die Funktionstasten F1 bis F8 intern ab Null durchnumeriert (F1=Funktionstaste Nummer 0, F8 (HELP)=Funktionstaste Nummer 7).

In Speicherzelle 239, den Tastaturpufferzähler, wird der Wert eins gepokt, da eine (!) gedrückte Taste simuliert werden soll. Der Befehl, mit der F1 belegt wurde, wird erst dann ausgeführt, wenn der Programmmodus mit END verlassen und in den Direktmodus zurückgekehrt wird.

Mit diesen Kenntnissen ist ein Verständnis der vorgestellten Routine nicht mehr allzu schwer. In Zeile 10000 wird der String K1$ definiert, mit dem F1 belegt werden soll. Beachten Sie bitte, daß die darin aufgeführten - unsinnigen - LIST-Parameter

(LIST 00000-00000) nur vorläufig Geltung besitzen und im weiteren Programablauf verändert werden.

Die Variable LS kennzeichnet den "LIST-Start", das heißt die erste auf dem Bildschirm auszugebende Programmzeile. Der anschließend erfolgende Sprung führt zu Zeile 10080, in der der String K1 gemäß dem zu listenden Bereich (momentan 100-200) verändert wird. F1 wird mit dem geänderten String

```
K1 (k1$="scnclr:list    0-    0"+chr$(13)+"goto 10030"+ chr$(13))
```

belegt. In Zeile 10110 wird die Länge des Funktionstastentextes in Speicherzelle 1373 gepokt, die Funktionstastennummer in 1374, und der Wert eins in den Tastaturpufferzähler (Speicherzelle 239).

Der Befehl END (Zeile 10120) bewirkt das Verlassen des Programmodus. Der simulierte Funktionstastenbefehl wird nun im Direktmodus ausgeführt. Der Bildschirm wird gelöscht, die Zeilen 100-200 gelistet und das Programm mit Zeile 10030 fortgesetzt.

In dieser Zeile wird auf eine Taste gewartet. Je nachdem welche der vier Cursortasten gedrückt wurde, wird LS, die erste zu listende Zeilennumer, entsprechend verändert. Anschließend wird F1 entsprechend dem neuen Wert von LS wieder umdefiniert, und der neue Programmbereich gelistet.

Dieses Programm soll selbstverständlich nur ein Beispiel zur programmgesteuerten Nutzung der Funktionstasten sein. Wofür Sie persönlich diesen Trick einsetzen, hängt allein von Ihrer Phantasie ab. Merken Sie sich bitte die allgemeine Vorgehensweise:

1. Funktionstaste mit gewünschten Befehlen belegen
2. Länge des Funktionstastentextes in 1373 poken
3. Funktionstastennummer in 1374 poken
4. Den Wert eins in 239 (Tastatuorpufferzähler) poken
5. Programmodus mit END verlassen

Beachten Sie bitte, daß der letzte auszuführende Befehl ein GOTO-Befehl mit jener Zeilennummer sein sollte, ab der das Programm fortgesetzt werden soll.

7.2. Graphiken abspeichern und laden

Im vorgestellten Malprogramm fehlen Routinen zum Abspeichern und Laden der erstellten Graphiken. Diese Routinen wurden absichtlich weggelassen und werden nun vorgestellt, und zwar aus folgendem Grund: Die vorgestellte Routine erlaubt nicht nur das Laden und Abspeichern des Graphikbildschirms, sondern beliebiger Speicherbereiche.

Sie können diese Routine daher nicht nur problemlos als Erweiterung in das Malprogramm einbauen, sondern in eigenen Programmen zum Beispiel auch zum Abspeichern des normalen Textbildschirms oder anderer interessanter Speicherbereiche verwenden.

Programmbedienung

Das folgende Programm verwendet zur Demonstration eine kleine Graphik, die in den Zeilen 180 bis 220 erstellt wird. Diese Programmzeilen können Sie jederzeit durch ein eigenes Programm zur Erstellung von Graphiken ersetzen.

Nach dem Starten des Programms mit "RUN" wird Ihre Graphik gezeichnet. Anschließend wird Sie unter dem Namen "GRAPHIK" auf Diskette gespeichert, der Graphikbildschirm wird gelöscht und die Graphik wieder von Diskette geladen.

Diese Anwendung ist selbstverständlich nicht allzu flexibel. Um beide Programmteile, das Laden und das Abspeichern einer Graphik, individuell in eigenen Programmen einzusetzen, gehen Sie bitte wie folgt vor:

1. "Hängen" Sie die Zeilen 10000 bis 10210 an das Ende Ihres eigenen Programms an.

2. Die beiden ersten Befehle des Gesamtprogramms müssen lauten

   GOSUB 10150
   NA$="GRAPHIK"

   Anstelle von "GRAPHIK" kann selbstverständlich ein beliebiger anderer Dateiname angegeben werden.

3. Abgespeichert wird eine Graphik mit: GOSUB 10000

4. Geladen wird die Graphik mit: GOSUB 10080

Mehr ist nicht erforderlich, um diese Routinen in beliebige Graphikprogramme einzubauen. Wenn Sie anstelle einer Graphik auch andere Speicherbereiche auf Diskette speichern beziehungsweise von Diskette laden wollen, lesen Sie bitte den Abschnitt über den Programmablauf.

Programmlisting

```
100 rem **********************
110 rem * save/load graphik  *
120 rem * (c) s.baloui, 1986 *
130 rem **********************
140 :
150 gosub 10150:rem datas einlesen
160 na$="graphik":rem filename
170 :
180 rem *** demo ***
190 graphic 1,1:rem graphik einschalten
200 for i=10 to 130 step 3
210 ::box 1,i*2,i,i*2+50,i+50
220 next
230 :
240 gosub 10000:rem save graphik
```

```
250 scnclr:char 1,1,10,"bitte taste druecken":getkey a$:rem auf
taste warten
260 gosub 10080:rem load graphik
270 getkey a$:graphic 0:end
280 :
290 :
300 :
10000 rem *** graphik speichern ***
10010 n$=na$+",s,w":ln=len(n$)
10020 open 1,8,15,"s:"+na$:close 1
10030 poke dec("0400"),ln:poke dec("0424"),ln:rem laenge uebergebe
n
10040 fori=1toln:poke i+dec("025d"),asc(mid$(ns$,i,1)):next
10050 sys dec("03f7"):rem aufruf 'save'
10060 return
10070 :
10080 rem *** graphik laden ***
10090 nl$=na$+",s,r":ln=len(nl$)
10100 poke dec("0400"),ln:poke dec("0424"),ln:rem laenge uebergebe
n
10110 fori=1toln:poke i+dec("025d"),asc(mid$(nl$,i,1)):next
10120 sys dec("041b"):rem aufruf 'load'
10130 return
10140 :
10150 rem *** datas ***
10160 data 169,2,162,8,160,2,32,186,255,169,11,162,94,160,2,32,189
,255,169,0
10170 data 133,216,169,32,133,217,169,216,162,0,160,64,32,216,255,
96,169,2,162,8
10180 data 160,2,32,186,255,169,11,162,94,160,2,32,189,255,169,0,3
2,213,255,96,-1
```

Programmablauf

Zum Verständnis des Programmablaufs benötigen Sie das nachfolgend abgebildete Listing der Maschinenroutine und Kenntnis über den Umgang mit sogenannten "logischen Dateien".

Logische Dateien dienen zur Datenübertragung an Peripheriegeräte wie zum Beispiel an ein Diskettenlaufwerk. Der OPEN-Befehl öffnet eine logische Datei, wobei folgende Parameter angegeben werden müssen:

1. Logische Dateinummer
2. Geräteadresse
3. Sekundäradresse
4. Dateiname

Um den Graphikbildschirm abzuspeichern, muß ebenfalls eine logische Datei geöffnet werden, der diese Parameter übergeben werden. Die drei ersten Parameter werden dem Betriebssystem beim Aufruf der Routine SETFLS ($FFBA) in den Prozessorregistern übergeben (Akku=Dateinummer; X=Geräteadresse; Y=Sekundäradresse).

Anschließend wird der Routine SETNAM ($FFBD) der Dateiname übergeben, wobei beim Aufruf dieser Routine der Akku die Länge des Dateinamens enthalten muß und im X- und Y-Register ein Zeiger (in der Form Low-/High-Byte) auf die Adresse übergeben wird, an der sich der Dateiname im Speicher befindet, in unserem Fall die Adresse $025E.

Im Betriebssystem existiert die Routine SAVE ($FFD8), mit der sich ein beliebiger Speicherbereich auf Diskette speichern läßt. Diese Routine kann nun aufgerufen werden, wenn Ihr zuvor Beginn und Ende des Speicherbereiches mitgeteilt wurden, der in unserem Fall (Graphikspeicher) bei $2000 beginnt und bei $3FFF endet:

1. In der Zeropage wird ein Zeiger auf den Beginn des Speicherbereichs in der Form Lowbyte/Highbyte abgelegt, in der vorgestellten Routine in $D8/D9.

2. Der Akkumulator wird mit der Adresse geladen, an der sich das Lowbyte dieses Zeigers befindet (im Beispiel mit $D8).

3. X- und Y-Register werden mit einem Zeiger auf das Ende +1 (!) des abzuspeichernden Bereichs geladen (X=Lowbyte/Y=Highbyte).

4. Die SAVE-Routine kann aufgerufen werden.

Die Routine zum Laden einer Graphik funktioniert sehr ähnlich. Zuerst werden wieder der Routine SETFLS die Dateiparameter übergeben und anschließend der Routine SETNAM der Dateiname.

Anstelle der SAVE-Routine wird die Routine LOAD ($FFD5) zum Laden eines Speicherbereichs verwendet. Wenn dieser Routine im Akku der Wert null übergeben wird, wird die Datei automatisch an jene Stelle geladen, an der sie sich beim Abspeichern befand (die Adresse wird beim Abspeichern in der Datei vermerkt), es ist daher nicht nötig, anzugeben, wohin die Datei geladen werden soll. Das Graphikbild wird automatisch ab Adresse $2000 geladen.

SAVE-Routine

```
. 03f7  a9 02      lda #$02     ;dateinummer 2
. 03f9  a2 08      ldx #$08     ;geraeteadresse 8
. 03fb  a0 02      ldy #$02     ;sekundaeradresse 2
. 03fd  20 ba ff   jsr $ffba    ;setfls aufrufen
. 0400  a9 0b      lda #$0b     ;laenge dateinamen
. 0402  a2 5e      ldx #$5e     ;adresse des datei-
. 0404  a0 02      ldy #$02     ;nach x/y
. 0406  20 bd ff   jsr $ffbd    ;setnam aufrufen
. 0409  a9 00      lda #$00     ;$d8/$d9=zeiger auf
. 040b  85 d8      sta $d8      ;den anfang des
. 040d  a9 20      lda #$20     ;graphikspeichers
. 040f  85 d9      sta $d9      ;($2000)
. 0411  a9 d8      lda #$d8     ;akku=zeigeradresse
. 0413  a2 00      ldx #$00     ;x=low von ende+1
. 0415  a0 40      ldy #$40     ;y=high von ende+1
```

```
. 0417  20 d8 ff          jsr $ffd8          ;save aufrufen
. 041a  60                rts                ;=> basic
```

LOAD-Routine

```
. 041b  a9 02             lda #$02           ;dateinummer 2
. 041d  a2 08             ldx #$08           ;geraeteadresse 8
. 041f  a0 02             ldy #$02           ;sekundaeradresse 2
. 0421  20 ba ff          jsr $ffba          ;setfls aufrufen
. 0424  a9 0b             lda #$0b           ;laenge filename 10
. 0426  a2 5e             ldx #$5e           ;adresse des datei-
. 0428  a0 02             ldy #$02           ;namens nach x/y
. 042a  20 bd ff          jsr $ffbd          ;setnam aufrufen
. 042d  a9 00             lda #$00           ;an ursprung laden
. 042f  20 d5 ff          jsr $ffd5          ;load aufrufen
. 0432  60                rts                  ;=> basic
```

Nun dürfte auch verständlich sein, wie mit dieser Routine beliebige Speicherbereiche abgespeichert und wieder geladen werden können. Die Parameter müssen geändert werden, die der Routine SAVE übergeben werden, der Beginn und das Ende des abzuspeichernden Bereichs. Hierzu können Sie entweder dieses Listing entsprechend geändert eingeben oder aber Sie ändern diese Parameter mit POKE-Befehlen individuell ab. In diesem Fall sind abzuändern:

1. $040A (Lowbyte des Bereichsanfangs)
2. $040E (Highbyte des Bereichsanfangs)
3. $0414 (Lowbyte des Bereichsendes+1)
4. $0416 (Highbyte des Bereichsendes+1)

7.3. Zeileninvertierung

Sowohl dieses als auch das folgende Programm dienen dazu, BASIC-Programmen einen professionellen "Touch" zu geben. Die Routine "Zeileninvertierung" erlaubt das blitzschnelle Invertieren

von bis zu sechs Bildschirmzeilen. Eine solche Invertierroutine wird in professionellen BASIC-Programmen häufig zur Hervorhebung bestimmter Bildschirmteile benötigt.

Ein Beispiel: Sie schreiben ein Dateiverwaltungsprogramm. Der Bildschirm soll als "Fenster" dienen, in dem immer ein Ausschnitt der kompletten Datei dargestellt wird, zum Beispiel ein Datensatz pro Zeile.

Gehen wir davon aus, daß der Benutzer die Datei beliebig "durchblättern" kann. In diesem Fall benötigt er selbstverständlich eine Markierung, die Auskunft über die momentane Position gibt. Es bietet sich an, die betreffende Zeile zu invertieren. Anstelle eines ein Zeichen großen Cursors wird somit ein "Zeilencursor" verwendet, durch den der aktuelle Datensatz hervorgehoben wird. Wie Sie sehen werden, ist diese Aufgabe kein Problem für die Invertierroutine.

Ein zweites Beispiel ist die Ausgabe einer Fehlermeldung auf dem Bildschirm. Um die Fehlermeldung hervorzuheben, könnte Sie blinkend dargestellt werden. Diese Möglichkeit bietet das BASIC des PLUS/4 zwar ebenfalls (Flash on/Flash off), die Blinkgeschwindigkeit können Sie jedoch nicht beeinflussen.

Mit einer Invertierroutine läßt sich das Problem eleganter lösen: Die Fehlermeldung wird in einer Schleife ständig invertiert. Da die Invertierung einer bereits invertierten Zeile zur Normaldarstellung führt, wird die Fehlermeldung abwechselnd invertiert und wieder normalisiert, sie blinkt.

Den zeitlichen Abstand der Invertierungen und damit die Blinkfrequenz können Sie mit Hilfe von Verzögerungsschleifen nach Belieben bestimmen.

Programmbedienung

Wenn Sie die Invertierroutine in eigenen Programmen anwenden wollen, müssen Sie die Zeilen 10000 bis 10060 des Demopro-

gramms an das Hauptprogramm "anhängen" (zum Beispiel mit der beschriebenen MERGE-Routine). Die Routine wird folgendermaßen aufgerufen:

1. Der erste Befehl des Gesamtprogramms muß lauten:
   GOSUB 10000
   Dieser Befehl liest die Datazeilen ein und damit die Maschinenroutine, die für die Invertierung verantwortlich ist.

2. Der Routine muß angegeben werden, wie viele Zeichen invertiert werden sollen (maximal 127 Zeichen), indem der entsprechende Wert minus eins in $0414 gepokt wird. Beispiel: POKE DEC("0414"),39 invertiert eine Zeile; POKE DEC("0414"),79 invertiert zwei Zeilen; POKE DEC("0414"),119 invertiert drei Bildschirmzeilen.

3. Die Routine wird mit SYS DEC("03F7"),LI aufgerufen, wobei LI die Nummer jener Zeile ist, mit der die Invertierung beginnen soll (1-25).

Beispiele:

1. poke dec("0414"),39:sys dec("03f7"),5 invertiert die komplette Zeile Nummer fünf.

2. poke dec("0414"),79:sys dec("03f7"),1 invertiert die erste und die zweite Bildschirmzeile.

3. poke dec("0414"),99:sys dec("037f"),2 invertiert die erste und zweite Bildschirmzeile komplett und die ersten 20 Zeichen der dritten Zeile.

4. pokedec("0414"),39:for i=1 to 25:sys dec("03f7"),i:next invertiert den kompletten Bildschirm.

Programmlisting

```
100 rem ***********************
110 rem * invertier-routine  *
120 rem * (c) s.baloui, 1986 *
130 rem ***********************
140 :
150 rem invertiert bis zu 127 zeichen
160 rem ab beliebiger bildschirmzeile
170 rem $0414 = zeichenanzahl minus 1
180 rem aufruf: sys dec("03f7"),li
190 rem wobei li=1.zu invertierende zeile
200 :
210 :
220 rem *** demo ***
230 gosub 10000:rem datas einlesen
240 scnclr
250 for i=1 to 24:print "dies ist ein test d.invert.-routine":next
260 poke dec("0414"),39:rem 39+1 zeichen=1 zeile invert.
270 for i=1 to 24
280 sys dec("03f7"),i
290 next
300 goto 270
310 :
320 :
10000 rem *** datas ***
10010 s=dec("03f7"):i=0
10020 read a:if a<>-1 then poke s+i,a:i=i+1:goto 10020
10030 return
10040 data 169,216,162,11,133,216,134,217,32,145,148,32,132,157,16
9,40,24,101
10050 data 216,133,216,144,2,230,217,202,208,242,160,79,177,216,73
,128,145,216
10060 data 136,16,247,96,-1
```

Programmablauf

Die Maschinenroutine besteht zwar nur aus wenigen Bytes, dürfte für Interessierte jedoch vor allem wegen der Verwendung

von zwei bereits besprochenen Routinen des BASIC-Interpreters (siehe "Die Routinen des BASIC-Interpreters") interessant sein. Das Listing der Routine finden Sie auf den folgenden Seiten.

Zu Beginn der Routine wird in den Speicherzellen $D8/$D9 ein Zeiger auf den Bildschirm untergebracht. Dieser Zeiger weist nicht exakt auf den Beginn des Bildschirmspeichers ($0C00), sondern eine Zeile (40 Zeichen) davor, das heißt auf die Position $0BD8. Den Grund werden Sie gleich erfahren.

Im nächsten Schritt werden nacheinander die Routinen des BASIC-Interpreters CHKKOM ($9491) und GETBYT ($9D84) aufgerufen. CHKKOM liest das dem SYS-Befehl und der Aufrufadresse - zum Beispiel SYS DEC("03F7"),12 - folgende Komma ein. GETBYT liest den dem Komma folgenden Ein-Byte-Wert aus dem BASIC-Text ein (wobei es sich auch um eine Variable handeln kann) und übergibt den Wert (im Beispiel 12) im X-Register.

Dieser Wert, der die Nummer jener Zeile darstellt, ab der die Invertierung beginnen soll, wird im folgenden als Schleifenzähler verwendet. Der Zeiger $D8/$D9 auf den Bildschirmspeicher wird genau "X"-mal (wobei X gleich dem Wert im X-Register ist) um 40 erhöht, das heißt um die Anzahl der Zeichen pro Zeile beim PLUS/4.

Wurde beim Aufruf der Routine als Startzeile zum Beispiel der Wert eins übergeben, wird $D8/$D9 genau ein Mal um den Wert 40 erhöht und weist damit exakt auf den Beginn des Bildschirmspeichers, das heißt auf die erste Spalte der ersten Zeile.

Die Zeichen ab der errechneten Adresse werden nun invertiert, indem sie mit $80 Exklusic-ODER verknüpft werden (man sagt auch, *geORt* werden). Bit 7 jedes Zeichens, das über die Invers- beziehungsweise Normaldarstellung entscheidet (Bit 7 gelöscht = normal; Bit 7 gesetzt = invertiert), wird durch diese Verknüpfung gesetzt, wenn es zuvor gelöscht war, und gelöscht, wenn es zuvor gesetzt war. Der Zustand dieses Bits wird "umgedreht". Invertierte Zeichen werden also normalisiert, und normal dargestellte Zeichen werden invertiert.

Wie viele Zeichen auf diese Weise "umgedreht" werden, hängt vom Wert der Speicherzelle $0414 ab, mit der das Y-Register zu Beginn der Schleife geladen wird. Als Standardwert ist $27 (=dezimal 39) eingestellt, was zur Invertierung von 40 Zeichen führt, also exakt einer Bildschirmzeile.

Invertierroutine

```
. 03f7  a9 d8       lda #$d8      ;zeiger auf bild-
. 03f9  a2 0b       ldx #$0b      ;schirmanfang
. 03fb  85 d8       sta $d8       ;minus eine zeile
. 03fd  86 d9       stx $d9       ;nach $d8/$d9
. 03ff  20 91 94    jsr $9491     ;chkkom
. 0402  20 84 9d    jsr $9d84     ;getbyt
. 0405  a9 28       lda #$28      ;entsprechend der
. 0407  18          clc           ;uebergebenen
. 0408  65 d8       adc $d8       ;startzeile $28
. 040a  85 d8       sta $d8       ;(dezimal 40) zu
. 040c  90 02       bcc $0410     ;$d8/$d9 addieren
. 040e  e6 d9       inc $d9
. 0410  ca          dex
. 0411  d0 f2       bne $0405
. 0413  a0 27       ldy #$27      ;$27 (dezimal 39)
. 0415  b1 d8       lda ($d8),y   ;+1 zeichen ab
. 0417  49 80       eor #$80      ;zeigeradresse
. 0419  91 d8       sta ($d8),y   ;invertieren
. 041b  88          dey
. 041c  10 f7       bpl $0415     ;fertig? nein =>
. o41e  60          rts           ;ja => basic
```

7.4. Komfortable Menüverwaltung

Der Trend in der Programmierung geht heutzutage immer mehr in Richtung auf möglichst komfortable "Benutzeroberflächen". In der Praxis ist damit zumeist die Verwendung von "Windows" und sogenannten "Pull-Up" oder "Pull-Down"-Menüs gemeint. Diese

Techniken sollen dem Benutzer vor allem die Auswahl bestimmter Funktionen aus sogenannten "Auswahlmenüs" erleichtern.

Wenn Sie das Malprogramm, die KFZ-Überwachung oder die integrierte Datei-/Textverarbeitung eingetippt haben, wissen Sie, was unter einem Auswahlmenü zu verstehen ist. Die in diesen Programmen verwendeten Techniken zur Anwahl eines bestimmten Kommandos bestehen in der Steuerung über die Funktionstasten (Malprogramm; Text-/Dateiverarbeitung) beziehungsweise in der Eingabe einer Zahl, die einer bestimmten Funktion zugeordnet ist (KFZ-Überwachung).

Diese Techniken erfüllen zwar Ihren Zweck, sind jedoch zweifellos nicht so "schick" und wirken weniger professionell als die genannten neueren Menütechniken.

Dieser Mangel wird mit der im folgenden dargestellten Routine zur Menüsteuerung behoben. Diese Routine können Sie in beliebige eigene Programme integrieren. Das Menü wird in einer oder auch mehreren Zeilen ab Zeile 21 auf den Bildschirm gemalt, zum Beispiel in folgender Form:

```
Eintragen Suchen    Aendern Loeschen Druck
Sortieren Speichern Laden   Ersetzen
```

Welcher Menüpunkt selektiert ist, wird dem Benutzer durch Invertierung des betreffenden Punktes angezeigt. Nach dem Aufruf der Routine wird immer der erste Menüpunkt (im Beispiel "Eintragen") invers dargestellt. Welcher Menüpunkt selektiert wird, kann der Benutzer mit Hilfe der Cursortasten steuern. Mit "CURSOR RECHTS" wird der nächste, mit "CURSOR LINKS" der jeweils vorige Menüpunkt selektiert und invers dargestellt. Mit der Taste "RETURN" kann der selektierte Menüpunkt "aktiviert" werden.

Programmbedienung

Um die Routine in Ihre eigenen Programme zu integrieren, müssen Sie in diese wie in den vorigen Programmen die Datazeilen (Zeile 10000-10090) eingeben. Bei der Benutzung der Routine sind folgende Schritte zu beachten:

1. Der erste Befehl des Gesamtprogramms muß lauten: GOSUB 10000

2. Das Menü, das sich über mehrere Zeilen erstrecken kann, muß ab Zeile 21 gezeichnet werden, wobei das erste Zeichen dieser Zeile ein Leerzeichen (" ") sein muß! Die einzelnen Menüpunkte müssen durch mindestens ein Leerzeichen voneinander getrennt sein.

3. In $0580 muß die Anzahl der Menüpunkte gepokt werden.

4. Nun kann die Routine mit SYS DEC("055F") aufgerufen werden.

5. Die Nummer des angewählten Menüpunktes befindet sich nach Anwahl eines selektierten Punktes mit "RETURN" in der Speicherzelle $D8 und kann mit PEEK(DEC("D8")) abgefragt werden.

Ein kleines Programmbeispiel hierzu:

```
100 gosub 10000:rem datas einlesen
110 char 1,0,21," eintragen suchen aendern loeschen":rem menue ab
zeile 21 (leerzeichen in spalte 1)
120 poke dec("0580"),4:rem 4 menuepunkte
130 sys dec("055f"):rem menueroutine aufrufen
140 print peek(dec("d8")):rem auswahl abfragen
```

Wie bereits erwähnt, wird ein bestimmter Menüpunkt mit den Cursortasten selektiert und mit "RETURN" aktiviert. Ich emp-

fehle Ihnen die Eingabe des Demoprogramms. Wenn Sie dieses starten, werden Sie sofort sehen, wie komfortabel und problemlos der Umgang mit dieser Routine ist.

Programmlisting

```
100 rem ****************************
110 rem * menueverwaltungsroutine *
120 rem *   (c) s.baloui, 1986     *
130 rem ****************************
140 :
150 rem *** uebergabeparameter ***
160 rem $0580 = anzahl der menuepunkte
170 rem $055F = startadresse d.routine
180 rem das 1.zeichen der 1.menuezeile
190 rem (zeile 21) muss space (' ') sein
200 :
210 :
220 rem *** demo ***
230 gosub 10000:rem datas einlesen
240 scnclr:rem bildschirm loeschen
250 m$=" eintragen suchen aendern loeschen druck sortieren speiche
rn laden ersetzen"
260 poke dec("0580"),9:rem 9 menuepunkte
270 :
280 char 1,0,20,"----------------------------------------":rem 40
bindestriche"
290 char 1,0,21,m$:rem menue ab zeile 21 malen
300 char 1,0,23,"----------------------------------------":rem 40
bindestriche"
310 sys dec("055f"):rem menueroutine aufrufen
320 print chr$(19):rem cursor home
330 print "sie waehlten den"peek(dec("d8"))"menuepunkt aus"
340 input "nochmal (j/n)";a$:if a$="j" then 240:else end
350 :
360 :
370 rem *** datas ***
10000 s=dec("055f"):i=0
10010 read a:if a<>-1 then poke s+i,a:i=i+1:goto 10010
```

```
10020 return
10030 data 160,1,162,1,32,195,5,134,217,132,218,32,228,255,240,251
,166,217,164
10040 data 218,201,13,240,59,201,157,240,25,201,29,208,235,192,9,1
76,231,32
10050 data 180,5,232,189,72,15,201,32,240,248,32,195,5,200,208,210
,192,2,144
10060 data 210,32,195,5,202,189,72,15,201,32,240,248,202,189,72,15
,201,32,208
10070 data 248,232,32,195,5,136,208,180,132,216,189,72,15,201,32,2
40,15,73,128
10080 data 157,72,15,232,208,241,134,217,32,180,5,166,217,96,-1
```

Programmablauf

Wie Sie bereits anhand der Länge des nachstehend abgebildeten Listings erkennen können, handelt es sich bei dieser Maschinenroutine um eine der komplexesten in diesem Buch.

Eine eingehende Erläuterung des Programms ist im Rahmen dieses Buches leider nicht möglich. Ich beschränke mich daher auf den prinzipiellen Programmablauf.

Der erste Menüpunkt wird invertiert. Anschließend wird auf eine Taste gewartet (GETIN = Tastaturabfrage, siehe "Betriebssystemroutinen"). Wurde "RETURN" gedrückt, wird die Nummer des aktuellen Menüpunktes in $D8 gespeichert und nach BASIC zurückgekehrt.

Wurde "CURSOR RECHTS" gedrückt, wird überprüft, ob der letzte Menüpunkt bereits erreicht ist, und der Befehl daher nicht durchgeführt werden darf. Ist dies der Fall, wird sofort zur Tastaturabfrage zurückgekehrt. Ansonsten wird der aktuelle (invertierte) Menüpunkt normalisiert. Die Routine überliest nun alle folgenden Leerzeichen und tastet sich zum nächsten Menüpunkt vor. Die Zeichen, aus denen dieser besteht, werden nacheinander invertiert, bis ein Leerzeichen an der zu bearbeitenden Position anzeigt, daß das Ende des Menüpunktes erreicht wurde.

"CURSOR LINKS" verläuft analog und wird nur dann ausgeführt, wenn nicht momentan bereits der erste Menüpunkt selektiert ist. "CURSOR LINKS" führt zuerst zur Normalisierung des aktuellen Menüpunktes. Die Routine tastet sich zurück bis zum vorhergehenden Menüpunkt, das heißt, sie überliest alle dazwischenliegenden Leerzeichen.

Das Programm tastet sich nun weiter zurück, bis der Anfang des neuen Menüpunktes erreicht wird, das heißt bis ein Leerzeichen eingelesen wird. Die Position des ersten Zeichens des nun aktuellen Menüpunktes steht damit fest, und dieser kann invertiert werden.

Menueverwaltungsroutine

```
. 055f  a0 01      ldy #$01     ;init menuenr.
. 0561  a2 01      ldx #$01     ;init zeiger
. 0563  20 c3 05   jsr $05c3    ;1.punkt invertier.
. 0566  86 d9      stx $d9      ;zeiger retten
. 0568  84 da      sty $da      ;menuenr. retten
. 056a  20 e4 ff   jsr $ffe4    ;getin aufrufen
. 056d  f0 fb      beq $056a    ;taste? nein =>
. 056f  a6 d9      ldx $d9      ;zeiger laden
. 0571  a4 da      ldy $da      ;menuenr. laden
. 0573  c9 0d      cmp #$0d     ;'return'?
. 0575  f0 3b      beq $05b2    ;ja =>
. 0577  c9 d9      cmp #$9d     ;cursor rechts?
. 0579  f0 19      beq $0594    ;ja =>
. 057b  c9 1d      cmp #$1d     ;cursor links?
. 057d  d0 eb      bne $056a    ;nein =>
. 057f  c0 09      cpy #$09     ;letzter menuepkt.
. 0581  b0 e7      bcs $056a    ;erreicht? ja =>
. 0583  20 b4 05   jsr $05b4    ;sonst punkt aktuel
. 0586  e8         inx          ;normal.u.menuenr.
. 0587  bd 48 0f   lda $0f48,x  ;inkrementieren
. 058a  c9 20      cmp #$20     ;bis zu naechstem
. 058c  f0 f8      beq $0586    ;punkt vortasten
. 058e  20 c3 05   jsr $05c3    ;und invertieren
. 0591  c8         iny          ;menuenr.inkrement.
```

```
. 0592  d0 d2       bne $0566     ;immer springen!
. 0594  c0 02       cpy #$02      ;1.menuepunkt
. 0596  90 d2       bcc $056a     ;erreicht? ja =>
. 0598  20 c3 05    jsr $05c3     ;sonst pkt.aktuell
. 059b  ca          dex           ;normalis.u.zeiger
. 059c  bd 48 0f    lda $0f48,x   ;vor punkt aktuell
. 059f  c9 20       cmp #$20      ;zu dem ende des
. 05a1  f0 f8       beq $059b     ;vorhergehenden
. 05a3  ca          dex           ;menuepunkts
. 05a4  bd 48 0f    lda $0f48,x   ;zuruecktasten,
. 05a7  c9 20       cmp #$20      ;dann zum ersten
. 05a9  d0 f8       bne $05a3     ;zeichen dieses
. 05ab  e8          inx           ;punkts zurueck
. 05ac  20 c3 05    jsr $05c3     ;menuepkt.invert.
. 05af  88          dey           ;menuenr.dekrement.
. 05b0  d0 b4       bne $0566     ;immer springen!
. 05b2  84 d8       sty $d8       ;menuenr.retten
. 05b4  bd 48 0f    lda $0f48,x   ;ab aktuellem
. 05b7  c9 20       cmp #$20      ;zeichen bis zu
. 05b9  f0 0f       beq $05ca     ;naechstem space
. 05bb  49 80       eor #$80      ;zeichen
. 05bd  9d 48 0f    sta $0f48,x   ;invertieren
. 05c0  e8          inx
. 05c1  d0 f1       bne $05b4     ;immer springen!
. 05c3  86 d9       stx $d9       ;zaehler retten
. 05c5  20 b4 05    jsr $05b4     ;punkt invertieren
. 05c8  a6 d9       ldx $d9       ;zeaehler holen
. 05ca  60          rts
```

Das große Grafikbuch zu C16/C116/Plus 4

Klar und ausführlich werden die Möglichkeiten der Grafik-Programmierung auf den Rechnern C16/C116/Plus 4 vorgestellt. Die Grafikbefehle werden erklärt, ihre Funktion wird an Beispielprogrammen, wie z. B. Spielen und Laufschriften, vorgestellt.
In die Programmierung von 2-D- und 3-D-Grafik wird eine Einführung gegeben, außerdem werden Grundlagen des CAD leicht verständlich vermittelt.
Anwendungen, z. B. der Rechnereinsatz bei der Auswertung statistischen Datenmaterials, zeigen die vielfältigen Möglichkeiten dieser Rechner.
Mit diesem Buch lernen Sie Ihren Rechner von seiner stärksten Seite kennen.

Löffelmann
Das große Grafikbuch C16/C116/Plus 4
ca. 300 Seiten, ca. DM 39,–
Erscheint ca. Juli/August

ISBN 3-89011-205-6

Das Maschinensprachebuch zu C16/C116/Plus 4

Programmierung in Maschinensprache – leichtgemacht.
Neben dem Befehlssatz des Prozessors wird die Verwendung eines Maschinensprache-Monitors vorgestellt.
Ein Vorzug dieses Buches ist die ausführliche und leichtverständliche Darstellung der wichtigsten Routinen, die das Betriebssystem zur Verfügung stellt, sowie deren Nutzung für die eigene Programmierung. An Beispielen wird der enorme Vorteil der Maschinensprache im Hinblick auf die Ausführungsgeschwindigkeit deutlich.
Tips und Tricks machen es möglich, den Rechner optimal auszunutzen.

Vüllers
Maschinensprachebuch C16/C116/Plus 4
ca. 300 Seiten, ca. DM 39,–
Erscheint ca. Juli/August

ISBN 3-89011-206-4

DAS STEHT DRIN:

Plus 4 Tips & Tricks ist eine Sammlung von Anregungen, Ideen und fertigen Lösungen zur Programmierung und Anwendung Ihres Plus/4.
Alles, was Sie brauchen, um eigene Programme zu schreiben.

Aus dem Inhalt:

- Hinweise zur Eingabe der Programme
- Verwendung der Datasette
- Anwenderprogramme
  Spiele
  Graphik
  Text- und Dateiverarbeitung
  Kfz-Überwachung
  Mathematik
  Harcopy
  Merge
  Shapeeditor
- Die wichtigsten Tips & Tricks
- Von BASIC bis ASSEMBLER
- Die wichtigsten ZEROPAGE-ADRESSEN
- Routinen des Betriebssystems und des BASIC-Interpreters
- Tips & Tricks für Fortgeschrittene

UND GESCHRIEBEN HAT DIESES BUCH:

Saïd Baloui stellt in zahlreichen Veröffentlichungen immer wieder seine langjährige Erfahrung im Umgang mit Commodore-Rechnern unter Beweis. Besonderes Interesse: Programmierung von Dateiverwaltungen.

ISBN 3-89011-203-X